Aalto University
School of Business

STREAMLINING ROBOTIC PROCESS AUTOMATION IMPLEMENTATIONS: THE ROLE OF MODULARITY

An Action Design Research

Master's Thesis
Vesa Saarikoski
Aalto University School of Business
Information and Service Management
Spring 2020

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Vesa Saarikoski

Title of thesis Streamlining robotic process automation implementations: the role of modularity

Degree Master of Science in Economics and Business Administration

Degree programme Information and Service Management

Thesis advisor(s) Esko Penttinen

Year of approval 2020 **Number of pages** 59 **Language** English

Abstract

This master's thesis focuses on studying how modularity can be applied in designing robotic process automation (RPA) solutions and what effects it has to the RPA implementations. Past research has found several benefits of increasing modularity in designing, coordinating, and managing complex systems and many kinds of design problems have been solved with it. This thesis examines how the principles of modularity resonate with RPA.

The study follows the Action Design Research method to examine a case in real-life context and to enable the participation of practitioners and end users in the case organization. It describes in detail how a modular RPA solution was created for automating an entity consisting of several different tasks in the case organization. The solution can be viewed as an artifact that can be utilized to calibrate an organization's RPA journey by adding efficiency in RPA projects. This is achieved by enhancing the scalability and reusability of the RPA components.

The results of this study suggest that RPA solutions can be viewed as complex systems into which modularity principles can be adapted, generating a variety of benefits from the viewpoints of automation development and administration. The main benefits include decreased implementation time and cost, enhanced error management, increased control of automation, increased capability to respond to versatile automation demand, and enhanced automation complexity management.

Keywords robotic process automation, component, modularity, scalability, reusability

Aalto-yliopisto
Kauppakorkeakoulu

Aalto-yliopisto, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Maisterintutkinnon tutkielman tiivistelmä

Tekijä Vesa Saarikoski

Työn nimi Modulaarisuuden rooli ohjelmistorobotiikan implementoinnin virtaviivaistamisessa

Tutkinto Kauppatieteiden maisteri

Koulutusohjelma Tieto- ja palvelujohtaminen

Työn ohjaaja(t) Esko Penttinen

Hyväksymisvuosi 2020 **Sivumäärä** 59 **Kieli** Englanti

Tiivistelmä

Tämä maisteritutkinnon tutkielma keskittyy tutkimaan, kuinka modulaarisuutta voidaan soveltaa ohjelmistorobotiikan (RPA) ratkaisujen suunnittelussa, ja mitä vaikutuksia sillä on RPA-implementaatioihin. Tähänastinen kirjallisuus on löytänyt useita hyötyjä modulaarisuuden lisäämisestä kompleksisten systeemien suunnittelussa, koordinoinnissa ja hallinnassa, ja sillä on ratkaistu useita erilaisia design-haasteita. Tämä tutkielma tarkastelee, kuinka modulaarisuuden toimintaperiaatteet resonoivat RPA:n kanssa.

Tutkielma seuraa suunnittelutoimintatutkimus (Action Design Research) -menetelmää tarkastellakseen tapausta tosielämän kontekstissa ja osallistaakseen eri työntekijöitä kohdeorganisaatiosta. Tutkielmassa kuvataan yksityiskohtaisesti, kuinka modulaarinen RPA-ratkaisu rakennettiin usean eri työtehtävän sisältämän kokonaisuuden automatisointia varten kohdeorganisaatiossa. Ratkaisua voidaan tarkastella artefaktina, jota voidaan hyödyntää kalibroimaan organisaation RPA-taivalta lisäämällä tehokkuutta RPA-projekteihin. Tämä saavutetaan parantamalla RPA-komponenttien skaalautuvuutta ja uudelleenkäytettävyyttä.

Tutkielman tulokset ehdottavat, että RPA-ratkaisut voidaan nähdä kompleksisina systeemeinä, joihin modulaarisuuden toimintaperiaatteet voidaan mukauttaa. Tämä tuottaa monia eri hyötyjä automaatioiden kehityksen ja hallinnan näkökulmasta. Pääasiallisia hyötyjä ovat implementaatioon käytetyn ajan ja sen kustannusten väheneminen, parantunut virheiden hallinta, parantunut automaation ohjaaminen, parantunut kyvykkyys vastata erilaisiin automaatiotarpeisiin ja parantunut automaation monimutkaisuuden hallinta.

Avainsanat ohjelmistorobotiikka, komponentti, modulaarisuus, skaalautuvuus, uudelleenkäytettävyys

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Background and motivation

In recent years, automating knowledge work has become a rising technology trend. Transferring the dull and repetitive tasks of knowledge work from humans to software robots is an intriguing opportunity to save costs for a variety of organizations, whether big or small. There are also numerous vendors offering Robotic Process Automation (RPA) services and the pace does not seem to be slowing any time soon as an increasing number of organizations handling processes in areas such as finance, public service and human resources are seeing the potential of RPA as a quick and inexpensive option for enhancing the effectiveness of business processes (Aguirre & Rodriguez, 2017). This has also created demand for research on how to make RPA projects as effective as possible.

The target organization for this research, The Finnish Tax Administration, is a large organization currently taking the first steps in their RPA journey having a few processes automated so far and several automations in development. The number of potential candidate processes to be automated is at least in hundreds. They are currently utilizing external RPA consultants to speed up the development of automations and support the advancement of their own RPA department. The plan is to match the development pace with the high automation demand and improve their own RPA capability to a point where they can run RPA as much in-house as possible.

To achieve these goals, redundant development work must be minimized, and the RPA implementations need to run as effectively as possible. This means that the RPA solutions must be made both flexible and effective at the same time. In addition, consistency in the RPA component development policies is required to avoid pointless frictions between the different implementation projects. Applying modularity to any kind of product system has a potential to empower scalability, consistency, reusability, and flexibility, but still preserve the valuable connections of the components and functions in them (Baldwin & Clark, 2000). Furthermore, there have been indications in the past research on for example software development by Subramanyam et al. (2012) that the flexibility and efficiency in modular development can be simultaneously achieved with scrupulous design choices

regarding the components and refactoring the component designs when necessary. The case organization is currently launching an RPA implementation project to automate part of their employee data management. This leads us to the focus of this study which is finding out how modularity can be applied to RPA solutions and what effects it has to the implementations in terms of efficiency.

1.2 Research questions, objectives, and scope

Modularity and its effects on different systems and environments have been widely studied in the past research. Some of the past studies include for example the works of Baldwin and Clark (1997), Schilling (2000), Salvador (2007), and Gamba and Fusari (2009). However, the principles and effects of modularity are yet to be comprehensively examined with RPA systems. Based on this research gap, the outlined motivation, and the known challenges in maximizing the effectiveness of RPA development projects mentioned by for example Rutaganda et al. (2017) and Boulton (2017), the thesis aims to answer the following questions:

1) How is modularity applied in RPA solutions?
2) What are the benefits of modularity in RPA implementations?

The goal of the research is to study the resonation between modularity and RPA. This happens by creating a modular RPA solution for managing employee data in the case organization that provides an understanding on how modularity can be used to add efficiency to RPA implementations. The impact on efficiency is examined not only from the viewpoint of the current project but also the potential future RPA projects in mind as the aim is for the future automations in the organization to be able to utilize as many parts of this solution as possible. The Finnish Tax Administration already had one automated process in action in the beginning of this project, in which a certain level of modularity could be seen. However, the concept of modularity has not been highlighted in the organization's RPA department and both its nature and significance might vary a lot in the minds of the RPA team members. The larger theoretical aim in this research is, on the hand to expand the existing literature on modularity by studying its effects on RPA, and on the other hand widen the spectrum of the relatively novel RPA-related research by connecting modularity to it.

The objective for the research is to be able to create a modular RPA solution by defining and analyzing a set of tasks related to the employee data management process in detail, and then, with the help of the analysis, automating those tasks with modular automation components. This will be done by organizing discussions and workshops with the business process experts, in other words, subject-matter experts (SMEs) of the employee data management to gather all the details related to it. The SMEs here are employees in the case organization who have profound knowledge of all the tasks related to the part of the employee data management that is being automated. Based on the information gathered from the SMEs, the automation logic is first designed and reviewed with the RPA experts. This is followed by cycles of developing, testing, and validating the RPA solution. Finally, the solution is launched into the production environment of the organization to run on its own. The final version will be a result of collaboration between the SMEs, different systems experts, and RPA experts.

The scope and results of the research help The Finnish Tax Administration and other organizations implementing RPA to calibrate their RPA journey and steer their automation implementations to the right direction. The principles of how modularity was used in this RPA solution and what benefits it produced can be used as practical examples for future implementations. In addition, many of the modular components created in this project have the potential to be utilized in other RPA automation solutions in the case organization as well. The organization has hundreds of business processes suitable for RPA and a potential to save huge amounts in costs. The benefits of streamlining the development projects right from the beginning can be remarkable.

1.3 Structure of the thesis

The case study in this research paper utilizes the Action Design Research (ADR) method which supports executing a practical empirical research inspired by a real-life situation (Sein et al., 2011). The outcome of this research is a modular RPA solution for automating a set of tasks related to employee data management in The Finnish Tax Administration. The thesis begins with a literature review that first describes the nature of RPA including its benefits and challenges, then conceptualizes modularity and studies its impact on complex systems, and finally links RPA to complex systems as the relationship between modularity and RPA is assessed.

The next part of the thesis presents the chosen research method and explains in more details why the method is suitable for this situation. In addition, the action steps taken during the project and the schedule of the project are discussed. The final topic in the method part includes evaluating the trustworthiness of the research.

After discussing the method, the study dives into the empirical part. First, the collection of data is described and the most important events during the project listed. This is followed by the first three of the four stages used in the ADR method. These stages are problem formulation, building, intervention and evaluation, and reflection and learning (Sein et al., 2011). The final part of the thesis is discussion where the fourth stage of the ADR method, formalization of learning, is presented in the form of design principles, theoretical contributions, and practical utility. Lastly, limitations of the study and future research suggestions are presented.

2 Literature review

2.1 Modularity

Modularity is a relatively old concept that has been studied by several researchers in the last decades. It can be summarized as "the building of a complex product or process from smaller subsystems that can be designed independently yet function together as a whole" (Baldwin & Clark, 1997). It is as a general property of a system as well as a strategy for organizing complex products and processes efficiently (Baldwin & Clark, 1997; Schilling, 2000). It can also be viewed as a technical or organizational characteristic, therefore being able to characterize both technological architectures and organizational structures (Tiwana & Konsynski, 2010). As a versatile concept, modularity can be used to solve many kinds of design problems and it has been proven useful in designing, coordinating, and managing complex systems (Baldwin & Clark, 2000; Ethiraj & Levinthal, 2004).

2.1.1 Hierarchical and complex system

Simon (1991) defines a hierarchical system as "a system that is composed of interrelated subsystems, each of the latter being, in turn, hierarchic in structure until we reach some lowest level of elementary subsystem". Almost all entities, whether they are biological, technological, or otherwise, can be viewed as hierarchically nested systems (Schilling, 2000). Schilling (2000) uses the term "component" instead of subsystem and notes similarly that each component of a system can be viewed as system of its own consisting of even finer components and so forth until we reach a point where the components are just "elementary particles" or until science limits the possibility to decompose any further.

Many of the systems around us are complex (Bar-Yam, 2019). A complex system can be for example a product or an organization (Ethiraj & Levinthal, 2004). Simon (1991) describes complex systems as follows: "Roughly, by a complex system I mean one made up of a large number of parts that interact in a nonsimple way. In such systems, the whole is more than the sum of the parts, not in an ultimate, metaphysical sense, but in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole." Thus, the complexity in this case originates primarily from the usually unknown nature and magnitude of interactions between different parts of the system and the system performance implications deriving from that (Ethiraj & Levinthal, 2004). Thus, to

understand the how a complex system works, we must have knowledge of not only how each part behaves but also how they act together to form the whole functionality of the system. The properties of a complex system can be viewed in for example the human nervous system (Bar-Yam, 2019).

2.1.2 Modular system

Baldwin and Clark (2000) describe modularity as follows: "Modularity is a structural fact: its existence can be determined by inspecting the structure of some particular thing. If the structure has the form of a nested hierarchy, is built on units that are highly interconnected in themselves, but largely independent of other units; if the whole system functions in a coordinated way, and each unit has a well-defined role in the system, then, by our definition, the thing is modular. This is true whether we are speaking of a brain, a computer, or a city." In other words, a modular system consists of units, or "modules", that are designed independently but function as an integrated whole (Baldwin & Clark, 1997). The benefits of modularity have been used for example in car production as the example below shows.

"*Carmakers, for example, routinely manufacture the components of an automobile at different sites and then bring them together for final assembly. They can do so because they have precisely and completely specified the design of each part. In this context, the engineering design of a part (its dimensions and tolerances) serves as the visible information in the manufacturing system, allowing a complicated process to be split up among many factories and even outsourced to other suppliers.*" (Baldwin & Clark, 1997)

When a system is viewed as interconnected or "nonmodular", the units or "parameters" are more strongly linked to each other and the whole structure can be seen as a single-module project where it is hard to change even a single parameter without having on effect on the others (Gamba & Fusari, 2009).

2.1.3 Modules and interfaces

A modular system component, "a module", can be described as "a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units" (Baldwin & Clark, 2000). In other words, a module can be defined by "a cluster of strongly interconnected parameters that are almost independent from the

parameters of other modules" (Gamba & Fusari, 2009). At the very minimum, a module is a portion of at least one product variant within a set of products (Salvador, 2007). Even though modules are structurally independent units, they work together in a larger system. Modules are linked together by "interfaces". An interface can be described as "a preestablished way to resolve potential conflicts between interacting parts of a design". Interfaces are elaborate descriptions of how modules fit together, communicate, and so forth, in other words, how they interact with each other. They are common information for the designers of the system to adopt. In order to minimize conflict, there must be specifications, meaning inputs and outputs, set for the interfaces. Together the interface specifications and modules form the architecture of the system. The architecture specifies what modules are part of the system and what their functions are (Baldwin & Clark, 1997). It enables both independence of structure and integration of function for the modules (Baldwin & Clark, 2000). In the carmaker example, a module can be viewed as one part of the car that is manufactured in a separate site. The possibility to manufacture the car in parts and assemble it in the end is enabled by the specified design of each part and how it fits together with other parts. This represents a concept of standardized interface that will be further explained later in this thesis. The system architecture in this example is formed by all the different parts and their functionalities.

A design can be defined as detailed description of a product. It is entirely determined by a number of parameters in it and their interconnections. The parameters are associated to one another when there is a physical or a logical connection or dependence between them. A modular design consists of a hierarchical set of modules tied together via specific "design rules". These design rules can be defined as "imperative principles of composition that each module must respect to maintain the compatibility with the other modules and the entire project". The hierarchical structure, acting as the framework for individual modules, designates them different structural functions based on their position (Gamba & Fusari, 2009). This "function binding" property of modularity will be assessed further later in this thesis.

2.1.4 Information visibility

Modularity can be achieved by dividing information into visible design rules and hidden design parameters. Modularity is beneficial only if this division is precise, unambiguous, and complete (Baldwin & Clark, 1997). The modules that are placed at the highest level of

the system can be called “hierarchical modules”. They pose a set of design rules or “visible information” for the dependent and connected modules that are placed at a lower level in hierarchy. These modules are called “hidden modules” (Gamba & Fusari, 2009). The visible information are decisions that affect subsequent decisions and concern the whole system. Ideally, the visible design rules are settled early in a design process and communicated widely to the involved parties (Baldwin & Clark, 1997). Visible information includes the interfaces of the modules, the system architecture, and standards (Baldwin & Clark, 2000). Standards are for testing if a module conforms to the design rules and how well it performs in comparison to other modules (Baldwin & Clark, 1997).

Hidden information or “hidden design parameters” only affect the piece of the system they are hidden in. The hidden elements can be chosen late and changed often. They also do not have to be communicated to anyone beyond the designers of the module (Baldwin & Clark, 1997). Information can be hidden by creating separate “abstractions” that hide the complexities of elements in the system. In other words, when the complexity of an element reaches a certain threshold, the system is broken apart into smaller blocks that have their own interfaces and design parameters. These design parameters become hidden information within those blocks and only affect that part of the system. Breaking the system into parts, in other words, “splitting”, is one of the six modular operators described later in this thesis. The goal in this division is to find the points where it is most natural to have a separation (Baldwin & Clark, 2000). On the one hand, the aim is to conserve the most productive interdependencies and connections, in other words, the functions, in the system. On the other hand, the goal is to enable innovative cycling and creativity, contained within the modules (Baldwin & Clark, 2000). In other words, the goal is to find the modularization where interdependencies between modules are minimized and the system is most cleanly decomposed (Langlois, 2002), which can be achieved by grouping strongly interacting elements or parts together and separate weakly interacting ones (Simon, 1991). This steers the system towards the optimal level of the unison of its flexibility and efficiency (Subramanyam et al., 2012). From project viewpoint, this logical division can be conducted for example by splitting a project based on different tasks or activities for individuals or groups which enables appropriate resources, technologies and/or coordination techniques to be applied on each task or activity. It also allows standardized service definitions, resource requirements, and choreographies for the

activities to be stored and reused during planning and analyzing phases of the future projects (Keith et al., 2013).

Hiding part of the information helps manage the complexity of the system because it splits the system into a set of independent smaller-sized elements while the design rules tie up the modules into a hierarchical structure, and thus prevents the progression towards the goal to be overwhelmed (Gamba & Fusari, 2009; Baldwin & Clark, 2000). In the end, if a significant amount of information relevant to subparts of the design is hidden within them, the system has been modularized and the system is represented in a less complex form to our mind (Baldwin & Clark, 2000). Considering for example a car, it could be natural to have a split between the production of the motor and the body. The motor experts would then work on the motor while the body designers construct the body. Each group has the detailed knowledge about their part to efficiently finish it, but do not need many details on the other team's work. The standardized interface specifications of the parts, known by both groups, would allow the smooth assembly in the end.

2.1.5 Modularity assessment perspectives

There are several different and overlapping viewpoints on how to assess modularity. Salvador (2007) gathers five common "definitional perspectives" of modularity from earlier studies. Those are component commonality, component combinability, function binding, interface standardization and loose coupling. In addition, a supportive aspect named "component separability" is discussed.

Salvador (2007) investigates modularity through a concept of product system building background for the concept as follows: "If we look at those products which are generally regarded as modular, such as computers, industrial machinery, Lego blocks, space stations, etc., we see that their common feature is that they can be easily configured in different ways. At the same time, to modify any of these products you do not generally have to change it completely, as you may just have to change a limited number of components. Accordingly, the goal of product modularity is to have different products while minimizing differences across their constituents." In relation to that example, Salvador (2007) defines the concept as follows: "Conceptually defining and empirically assessing product modularity implies that we know or hypothesize the variety of possible configurations as well as the way they are obtained. In other words, you cannot determine how easily you

can reconfigure something, if you do not know what the desired new configurations are. A single product, therefore, is not the appropriate unit of analysis for product modularity. Modularity will necessarily be a property of a set of products, namely a product system."

The first perspective, component commonality, can be measured by the number of standard components in a system, in other words, how many times a component is used in different designs. Shaftel (1972) defines it as "the choice of the number of standard designs or modules to be produced, how many of each part is to be included in each of the various designs, and how many of each standard design should be used in each application". From the viewpoint of the second perspective, component combinability, modularity is at a high degree when a variety of design configurations can be achieved by mixing and matching components from a given selection. In other words, the aim is to maximize the number of different possible configurations from a given number of parts. Furthermore, to allow the modules to be combinable with each other and achieve different product configurations, the modules must be separable. Separability indicates that a product variant can be built by first building its modules and then assembling them into a final product configuration. The separability of a component can be assessed by product disassembly as well because a separable component also must be easily detachable from the final product configuration where it was previously included. (Salvador, 2007)

The third perspective relates modularity to functions. From this viewpoint modularity entails a design of different components that execute a variety of overall functions through the combination of distinct building blocks or modules. Since the various design-specific functions are fulfilled by a combination of modules, the modules transform into "function modules" that are distinct parts of designs. Thus, a module can be defined as any distinct portion of a design (Salvador, 2007). From the fourth perspective, interface standardization, modularity in a system can be increased by creating a high independence between component designs enabled by standardizing the interface specifications, that is, inputs and outputs of components (Sanchez & Mahoney, 1996).

Interface standardization, shortly discussed before, is closely linked to both component combinability and the fifth perspective, loose coupling, and can practically be viewed as an enabler for them. In modular design, the standardized interfaces between components are determined for allowing a range of variations in components to be substituted into a system

architecture. Modules are components whose interface characteristics are within the range of variations allowed by the architecture. The modular architecture is flexible as design variations can be leveraged by substituting different components into the architecture without having to redesign other components. In other words, standardizing the interface specifications, or “interface design parameters” enables components to become both combinable and loosely coupled, since they can be effectively coordinated simply by demanding all the components to comply with the design rules, in this case, the standardized component interface specifications. This loose coupling of component designs in a modular architecture enables the mixing and matching of modules to produce a potentially high number of variations to the overall design with different distinctive functionalities, features, and/or performance levels. (Sanchez & Mahoney, 1996)

As we can see, all the dimensions overlap with each other to a certain degree. Schilling (2000) effectively links together all the perspectives defining modularity as “a continuum describing the degree to which a system's components can be separated and recombined” and states that the continuum “refers both to the tightness of coupling between components and the degree to which the "rules" of the system architecture enable (or prohibit) the mixing and matching of components”. The looseness or tightness of the coupling between the components in a design depends on how much a change in one component’s design requires compensating changes in the designs of other components (Sanchez & Mahoney, 1996). Whether it is loose or tight, all systems include a certain degree of coupling between components and only few systems have completely inseparable components that cannot be recombined. Thus, almost all systems have some degree of modularity (Schilling, 2000).

2.1.6 Modular operators

Increasing modularity of a design widens the scale of different modification possibilities for it. Baldwin and Clark (2000) talk about “modular operators” describing them as “dynamic possibilities that are inherent in modular structures”. In other words, they illustrate the changes that can be envisioned in a modular design to modify the existing structures into new structures in defined ways. These operators include splitting, substituting, augmenting, excluding, inverting, and porting. They go hand in hand with the assessment perspectives discussed in the previous chapter.

Splitting is a central option for a modular design because it allows a set of independent modules to be formed from an interconnected design or module by breaking it apart. In the case of splitting an interconnected design, the dependencies among the parameters must first be investigated to find the rational points where to conduct the splitting (Gamba & Fusari, 2009). An example of splitting presented by Gamba and Fusari (2009) provides an illustration to the matter.

"*A bank is considering specializing its business activity. One way to achieve this result is to split its business, currently interconnected as a single module, into a set of main functions (private investments, retail, small businesses, large businesses, etc.). That requires the choice of the target market and the creation of a set of independent modules/divisions based on the main functions. These modules are linked to a central decisional unit, which dictates the global business plan (design rules). The benefit of this structure is that each business unit is free to evolve, within the design rules, independently of what happens to the rest of the system. This can be made formal and effective by creating a pyramidal group, where the parent company (the central decision unit) controls the subsidiaries (divisions).*"

The substitution operator, in turn, enables an existing module or an interconnected design to be changed with a new one. When a module in the higher level in hierarchy is replaced, a "visibility cost" incurs because the lower level modules connected to it are affected by the design rules it poses on them. In other words, a new interface must be defined for these lower level modules. On the other hand, if a module in the lower level of that structure would be replaced, the internal differences in the new module would not require modifications in other components at the same level. This leads to design editing being more favorable in the lower levels as substitutions there have a limited impact on the design structure and a slower rate of change for higher-level modules. The flexibility to enhance an existing module without having to redesign the entire structure is one of the most important motivations to modularize a system (Gamba & Fusari, 2009). An example of substitution is presented below by Gamba and Fusari (2009).

"*The CPU is a hierarchical module: when it is changed, we usually also have to change the motherboard, which is a lower-connected module. On the other hand, it is possible to improve (applying the substitution operator) the video performance of a computer using a*

new graphics card, without affecting the global design structure. Hence, the graphics card can be considered a hidden module in the broad computer structure."

The augmenting operator is applied to either create a new level of hierarchy or increment an existing layer of modules. It improves a design by adding one or more modules to it without changing the existing design rules. The possibility to improve a design this way without the need of changes in the rest of the structure is another important motivation for modularization. Augmenting is often used together with the excluding operator that allows creating a minimal design with the opportunity of incrementing and increasing its size, scope, and depth later. The exclusion operator can provide both strategic and financial benefits. Strategically, the initial exclusion of a module mitigates the effects of potential failure of the whole design. Financially, the initial exclusion of a module from the system may allow to finance the subsequent expansion with the cash flows generated by the minimized initial design already operating. An illustrative example is shown below. (Gamba & Fusari, 2009)

"*An electricity company is planning to expand its production capacity by building a new nuclear power plant. It can follows two alternative approaches: in the first, one large production unit is built; the second approach is modular, because it comprises the construction of a series of lower-size power production units over time. Assume that the electricity price is the main driver of the decision. When the electricity price is highly volatile, the modular approach allows us to reduce risk and to shorten the time of the initial investment. This approach corresponds to applying the exclusion operator to the initial design and then using the augmenting operator within the design rules set at the beginning. An initial power plant of reduced size is constructed, with the option to expand (i.e., to augment) its capacity later should the economic conditions turn favorable.*"

Inversion means creating a new source of visible information by isolating the common properties embedded in different modules. It usually happens in three phases where the similarities in the modules are first detected, then the modules containing the similarities are split to single out the similar components, and finally a new module for the similar components is created and placed at a higher level in hierarchy (Gamba & Fusari, 2009). An example of inverting is presented below (see also Figure 1) by Gamba and Fusari (2009).

"*As an illustrative example, we can think of a merger between two auto manufacturers. Because the two companies belong to the same business area, their internal structures can potentially have some similar functional units or modules (e.g., the administrative department, the research department, or the production line of some common part of the vehicles). In the left part of Figure 7, Module 1 and Module 2 represent the production systems of the two companies involved in the merger. They have a common unit so the owner can apply the inversion operator to benefit from scale economies. First she has to isolate (i.e., split, as in the second step in Figure 7) the common components I in each firm, say, two lines that produce the same part for the vehicles of the merged companies, assuming the original brands are maintained. Next, a new module ˆI (i.e., a new production line) is designed, which can work with both original systems. Finally, the new module is placed on top of them by inverting its ranking in the hierarchy of the original design, as in the third step of Figure 7. The resulting company has only one production module that provides its services to the remaining modules of the two merged companies (i.e., Module 1 and Module 2 specific components).*"

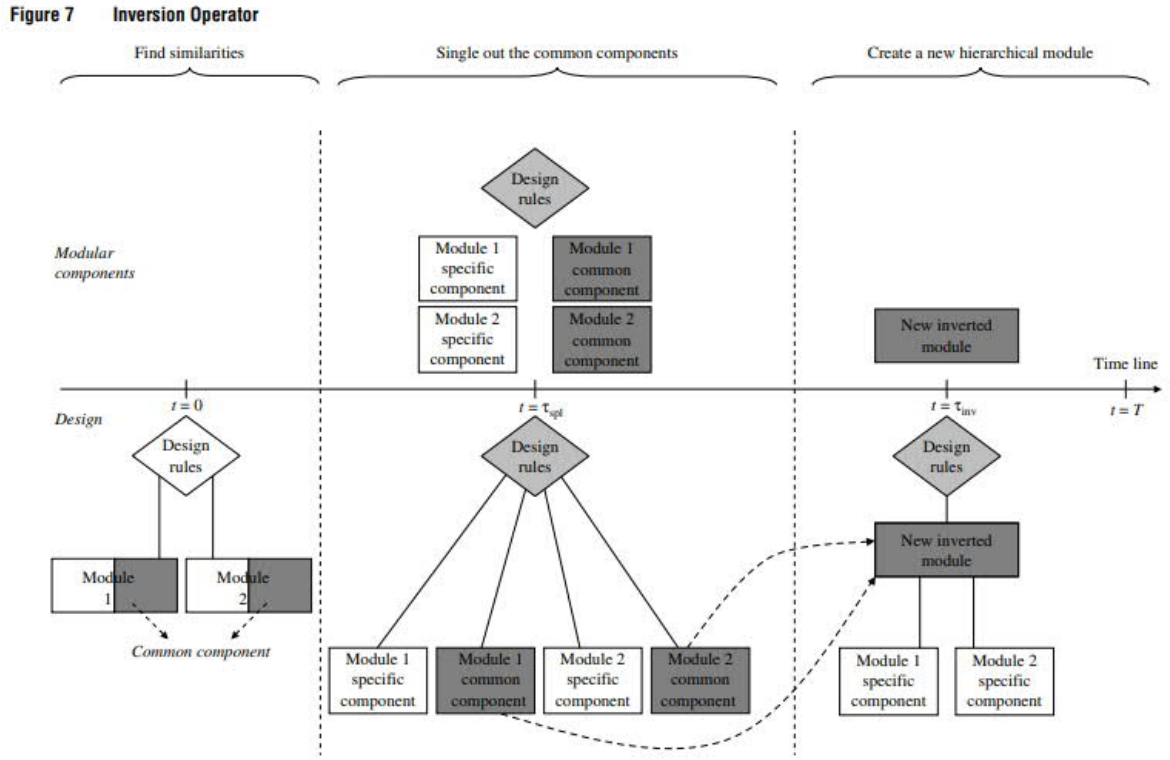

Figure 1. Inversion operator (Gamba & Fusari, 2009).

The sixth operator, porting, enables the creation of a module component that is compatible with other designs and structures, meaning that the module has an independent set of parameters that can serve well also out of the current design rules, and can thus be linked to other designs (Gamba & Fusari, 2009). In other words, porting is like inversion, but with the difference that portable modules are not trapped by the design rules of a particular system. Instead, they are free to drift from a system to system (Baldwin & Clark, 2000).

Of all the operators, splitting and substituting can be applied to both modular and nonmodular designs, the other four can only be used in a modular design. After creating a modular system, it can be improved upon by the six operators. Furthermore, all the operators can be applied locally to the system without interfering the other parts of the structure. The usage of the operators can be described as follows: "The life of a modular design can be divided into two typical phases: in the first phase, an interconnected design is turned into a modular one by splitting it; in the second phase, the design can be improved upon by further splitting, augmenting, replacing, porting, and excluding the existing modules. However, additional changes can be made to increase the value of the system. Among these, there is the possibility to improve the design by grouping similar or common functions that are spread across the structure into a single module. This module is then connected to all the other modules where the common function was present." (Gamba & Fusari, 2009)

2.1.7 Benefits of modularity

Enhancing modularity offers a variety of benefits related to the design. Due to the increased number of possible configurations that can be achieved with the given set of inputs, the flexibility of a system is improved (Schilling, 2000). This can be molded into strategic flexibility because the organization's capability to respond faster to changing markets and technologies by quickly producing product variations with new combinations of new or existing modular components is improved. With the help of the enhanced ability to produce new product variations, they can conduct continuous change more efficiently (Sanchez & Mahoney, 1996). Modularity also generates scale advantages in production. If there are common parts used in a variety of items, economies in part sourcing may significantly decrease the production costs. This also empowers the potential of "mass customization" as a high degree of variety can be achieved at a cost comparative to mass production (Baldwin & Clark, 2000).

One of the key advantages in modularization is the increased manageability of complexity. When the number of steps rises in an interconnected design, the difficulty to successfully complete the project increases. Furthermore, the project takes an increasing amount of time and the quality of the output may suffer. Limiting the scope of interaction between elements or tasks, and thus reducing the amount and range of potentially unproductive cycling occurring in a design or production process leads to better manageability of the process, less time spent on the process, increased probability of success, and enhanced quality of the final output (Baldwin & Clark, 2000). By enabling a structured representation of interdependencies through standardized interfaces, modularity also reduces the need for close coordination and makes workload more predictable (Susarla et al., 2010). Another modularity aspect that leads to saved time is the ability to work on different parts of a design concurrently as the independent blocks if a modular structure can all be developed simultaneously. In a situation where creating the design rules and conducting integration and testing do not require a massive amount of time, a modular task division cuts the time of completing a process. Combining the effects of concurrent development and reduced time in the cycles, the saved time might be substantial (Baldwin & Clark, 2000). In addition, modularity also improves specialization in the design process. As each module may evolve independently of the other modules within given design rules, each module can be worked with no worry of damaging the whole project (Gamba & Fusari, 2009).

Another benefit of modularity is enhanced risk mitigation and controllability of resources. The encapsulation feature of modularity enabling the information hiding helps prevent sensitive information being leaked and overexposed at an unwanted scale (Xue et al., 2013). This property to hide information also accommodates uncertainty in a design because the hidden parameters are isolated from other parts in the design. These parameters can vary, making them uncertain, but they only impact the block they have been isolated in, in other words, a module. This leads, again, to modular structures also being flexible as potential new knowledge being applied to a hidden module does not require much changes to the rest of the system. A new solution, having possibly various new components substituted, can thus be relatively simply incorporated with little loss of functionality (Baldwin & Clark, 2000; Schilling, 2000).

From a managerial viewpoint, the biggest advantage of modularity is decentralization, meaning that each module can be designed, made, and eventually implemented by a specific unit (Gamba & Fusari, 2009). This enables loosely coupled, flexible, 'modular' organizational structures. Enabling coordination with fully specified and standardized component interfaces potentially reduces the need for conducting managerial authority across the interfaces of organizational units developing components, which reduces the intensity and complexity of a firm's managerial job in product development, allowing a greater flexibility to launch a greater number and variety of product creation projects (Sanzhez & Mahoney, 1996).

2.1.8 The tradeoff of modularity

The effect of increasing modularity is to enable heterogeneous inputs to be recombined into various heterogeneous configurations. The pressure of a system to become more modular stems from how separable the components of the system are and the need to produce multiple configurations from diverse potential inputs. In other words, in systems where recombination is possible, there may be some combinations of particular components that work better together than others. By optimizing the components that work in a particular configuration, these valuable combinations achieve a functionality not achievable through combination of more independent components. The level of achieving this greater functionality with specific relations of components to one another can be called synergistic specificity of the system. In other words, the “customized” combination of components obtains synergy through the specificity of individual components to a certain configuration. Systems with a high level of synergistic specificity have a potential to accomplish things that more modular systems have cannot, but, with the price of decreasing level of recombinability (Schilling. 2000). However, this tradeoff is not always linear from development viewpoint. Results have been presented in for example a mass-customization software development research by Subramanyam et al. (2012) that threshold levels of modularity can be achieved where “the benefits of lower customization effort might not be neutralized by the loss of efficiency (increased defects and development effort)”. This is enabled by careful design choices and complementary measures such as refactoring (Subramanyam et al., 2012).

As mentioned before, the aim of modularity is to enable heterogeneous inputs to be recombined into various heterogeneous configurations. The more heterogeneous the inputs,

the more possible configurations are achieved through the recombinability. Furthermore, the more heterogeneous the demands made of the system, the more valued that recombinability is. A higher number of potential configurations of a system has a better potential to match the heterogeneous demands made of the system. The modularity of the system increases, decomposing the system into a group of modular components at ever-finer levels, until a balance is found between the pressure to become more modular and the functionality achieved through synergistic specificity. Because systems are typically nested hierarchies, each of these components is often a system of other components, facing its own balance between modularity and synergistic specificity. This trajectory might continue until we a level is reached at which the system is "relatively inseparable, is composed of relatively homogeneous inputs, or faces relatively homogeneous demands, or some combination of these" (Schilling. 2000). In other words, the circumstances determine that the benefits of modularization are not worth the cost. This might be the case for example for a system whose environment rarely changes (Langlois, 2002).

2.2 Robotic Process Automation

Robotic Process Automation as a term covers all tools operating on the user interface of other computer systems mimicking human activity (van der Aalst et al., 2018). The goal is to replace people by automating tasks, used to be done by humans, in an "outside-in" manner accessing the information systems through the presentation layer which means that their programming logic remains untouched (van der Aalst et al., 2018; Willcocks & Lacity, 2016). In other words, RPA operates on top of a company's information technology infrastructure rather than inside it (Institute for Robotic Process Automation, 2015). Gartner describes RPA as follows: "RPA tools perform [if, then, else] statements on structured data, typically using a combination of user interface interactions, or by connecting to APIs to drive client servers, mainframes or HTML code. An RPA tool operates by mapping a process in the RPA tool language for the software robot to follow, with runtime allocated to execute the script by a control dashboard." (Tornbohm, 2017).

Because the software robots, configured with the RPA tools, can communicate across IT systems via front-end instead of back-end like traditional software, it becomes possible for RPA to be integrated with basically any software used by a human, regardless of whether it is open to a third party integration (Asatiani & Penttinen, 2016). However, there are some characteristics that are required from the processes for RPA implementation to be

successful. For example, they usually need to be clearly defined, include rules-based routine tasks without subjective human judgement required, have structured data and deterministic outcomes (Aguirre & Rodriguez, 2017; Asatiani & Penttinen, 2016). The most suitable processes for RPA usually also have high transaction volumes (Lacity et al., 2015). However, the volume does not always need to be high if the process is otherwise business critical (Slaby, 2012). The tasks in automatable processes often include tipping, coping, pasting, extracting, merging, and moving data from one system to another (Aguirre & Rodriguez, 2017). Some examples of processes where RPA has been utilized are validating the sale of insurance premiums, generating utility bills, paying health care insurance claims, keeping employee records up to date (Lacity & Willcocks, 2017).

Some typical pitfalls in RPA implementations include for example not planning the RPA development project well. This might lead to the automations working too slow, being too expensive, and/or present too much complexity. In some cases, the robots have not been taken to use at all. (Rutaganda et al., 2017) An implementation might fail also because of poor managing of design and change. Rush to get robots deployed, companies sometimes overlook communication exchanges between the bots, which can break a business process. (Boulton, 2017)

2.2.1 RPA vs BPM

RPA can sometimes be confused with Business Process Management (BPM). There are, however, distinct differences between the two. RPA is for example easier to configure compared to BPM solutions as developers do not need any programming skills to set it up. RPA is a so called “lightweight” IT option meaning that does not interfere with the underlying computer systems and it can be applied more outside the IT department control than BPM tools (Lacity et al., 2015). Lightweight IT is typically considered cheap and easy to use. It often characterizes as a mobile technology that can generally be deployed without IT specialists (Bygstad, 2017). BPM solutions, in turn, suit better for heavyweight IT projects driven by IT professionals that include software engineering and adding code to a system (Bygstad, 2017; Willcocks et al., 2017). RPA also does not create a new application or a platform and does not store any transactional data which means a data model or database is not required unlike with BPM systems (Willcocks & Lacity, 2016). All in all, RPA can be viewed as a complementary option to BPM rather than a replacing one as each suit for different types of processes. BPM solutions should be used with IT-

owned processes that require IT expertise on high-valued investments such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems while RPA fits in so called “swivel chair” processes that are owned by operations and can be automated with the help of operations personnel using their business and process expertise (Lacity et al., 2015). Many of these swivel chair processes are back office processes such as accounts payable, accounts receivable, billing, travel and expenses, fixed assets, and human resource administration (Aguirre & Rodriguez, 2017).

2.2.2 Benefits of RPA

In the past few years, RPA solutions have seen a steep rise in demand which has naturally also resulted in increasing amount RPA vendors. The promise of RPA to quickly cut costs, link legacy applications together and achieve fast return on investment is appealing to many organizations (van der Aalst et al., 2018). In opposition to for example the long implementations and fuzzy business cases usually attached to business process management (Le Clair, 2017), achieving quick wins with little investment makes RPA an increasingly popular development option (van der Aalst et al., 2018). RPA is also very flexible and versatile compared to traditional software development. While major coding knowledge is usually required to make any major modifications to the operating logic of traditional software, the instructions for software robots can be modified through fairly simple logical statements, screen capture of the process executed by a human, or graphical process charts (Asatiani & Penttinen, 2016).

From employee viewpoint the core benefit of RPA is that it reduces the burden of doing repetitive and simple tasks day in, day out (Aguirre & Rodriguez, 2017). Hence, employees can practice more creativity, emotional intelligence and problem solving, and shift their focus from the dull tasks to more interesting, strategic, and productive projects (Willcocks et al., 2017; Asatiani & Penttinen, 2016). Many of those routine, non-core tasks have traditionally been good candidates for offshore outsourcing, especially the ones with a high full-time equivalent (FTE) value. Nowadays, RPA provides an alternative to gain the same benefits such as reducing staff costs and keeping focus on core operations. Not only can software robots work 24 hours a day for free pushing the cost savings even further, the traditional outsourcing challenges such as hidden cost of management, communication problems and complicated service level agreements can also be avoided. Tasks like invoice processing and data entry are now automated in increasing numbers

rather than outsourced to low-cost destinations, thus avoiding the possible backlash of from sending jobs abroad. In fact, RPA itself has a potential to create jobs as the automation projects often require management of robots, consulting, and analytics (Asatiani & Penttinen, 2016).

As previously mentioned, software robots can operate fully through the graphical user interface (GUI) leaving IT systems unaltered, which creates flexibility and speed to the implementation. This is a notable advantage compared to automation via back-end integration that usually calls for frequent and substantial redesigning of the existing system, making it slower. However, RPA is considered as more of a temporary solution that "fills the gap between manual processes based on legacy IT systems and redesigned processes running on fully automated systems" (Asatiani & Penttinen, 2016). In addition to cost reduction that is mainly based on productivity improvement, RPA implementations produce instant process-related benefits such as increasing process speed, error reduction (Aguirre & Rodriguez, 2017). They also help exposing the potential inefficient parts of the processes which can then be streamlined (Institute for Robotic Process Automation, 2015). Some indirect benefits have also been linked to using RPA like for example increased customer satisfaction, better regulatory compliance, increased consistency, getting products and services faster to market and increased employee skills and recognition (Willcocks et al., 2018).

3 Research method

The opportunity for a research was noticed by one the researchers during the first stages of the automation project in the target organization. The details of the research were then discussed with managers in the organization and the relevance of the study was agreed upon. The automation project started in June 2019 and carried on until May 2020 when the automation solution was deemed ready by the RPA team and scheduled to run on its own in the organization's production environment.

3.1 Background for choosing the research method

The project in this study was aimed at creating an RPA solution that can be used as an example for future RPA implementations in the organization on how to apply modularity in RPA. The target organization had only automated one business process at the time of launching this project and lacked experience on using modularity in RPA components. Each developer in the case organization had received a quick overview of the best practices for RPA development in short RPA courses, where some of the modularity principles are addressed, but in-depth explanations and understanding of the effects of those best practices were lacking. There was an agreed need for practical examples to support the knowledge gained through the initial RPA training and guide the automation endeavors to the right direction from the beginning.

The Finnish Tax Administration is planning to get full potential out of RPA during the next years. They have currently over 5000 employees and an endless amount of repetitive and rules-based processes that can be transferred from humans to robots to enhance process speed, reduce costs and release employees to more meaningful tasks. The aim is to build their own in-house RPA capability to answer this need for automation. This aim is to be supported by creating automations of quality in the beginning of the RPA journey and learn the best ways to develop RPA solutions in practice with the help of external RPA consultants. These first automations can then be leveraged by using them as model examples for future RPA implementations and by utilizing their modular parts in other automations. This is seen as a vital part of practical learning and getting the full-scale benefits out of RPA in the early stages by the organization.

This topic is tightly connected to the Information Systems (IS) field and the organizational context has a significant role in shaping the automation solution being produced. Implementing RPA is also a highly iterative process, combining observations, information, and knowledge from the several participants. It was also logical to divide the project into two parts, first mapping the tasks to be automated, and then applying this detailed knowledge about the tasks in creating an automation solution. This led to choosing the ADR methodology for this study which combines the principles from Design Research (DR) and Action Research (AR) (Sein et al., 2011). ADR can be defined as "a research method for generating prescriptive design knowledge through building and evaluating ensemble IT artifacts in an organizational setting" (Sein et al., 2011). In other words, it offers a chance to solve a real-life problem and emphasizes the value of organizational relevance over technological rigor (Rogerson & Scott, 2014). The nature of ADR suits well with the situation in the case organization as an RPA implementation is a very design-centered activity and creating an RPA related artifact supports the goal of creating in-house RPA knowledge in the organization. ADR method also supports the way that the automation solution, the final artifact, was produced, as a contribution between several different experts in the target organization and external RPA consultants.

3.2 Action Design Research

The research method in this study, ADR, is relatively new in the IS area. It "conceptualizes the research process as containing the inseparable and inherently interwoven activities of building the IT artifact, intervening in the organization, and evaluating it concurrently" (Sein et al., 2011). The method can be divided into two main challenges. The first challenge is "addressing a problem situation encountered in a specific organizational setting by intervening and evaluating" (Sein et al., 2011). In this case it was the need to create an automation solution of which parts can be utilized in future projects as much as possible and that can be used as an example of how modularity is applied to RPA components. The second challenge is "constructing and evaluating an IT artifact that addresses the class of problems typified by the encountered situation" (Sein et al., 2011). The final artifact that in this case is the automation solution, is produced by examining, evaluating, and building design knowledge in an organizational context. It reflects both the theoretical precursors and the aim of the researchers, and the influence of users and ongoing use in the organizational context (Sein et al., 2011). In addition, four design

principles were formed as a result of the project, adhering the ADR schema. These principles were formed from notations made while reviewing and improving the artefact.

The ADR is conducted in four different stages that have seven principles divided between them. These are presented in Figure 2 below. The first stage is problem formulation that presents the research problem in an organizational context. The second stage includes building, intervention, and evaluation, where the artefact goes through a number of intervention cycles. These cycles involve all the teams and participants of the ADR study as the artefact is constructed, reviewed, and improved towards its final version. The third stage presents points of reflection and learning from the project. Finally, the fourth stage, formalization of learning, presents generalized outcomes of the study in the form of design principles, practical utility, and theoretical contributions (Sein et al., 2011). The empirical research of this case study bases on these stages and principles and is documented to follow this structure with the exception that the formalization of learning is placed in the discussion chapter.

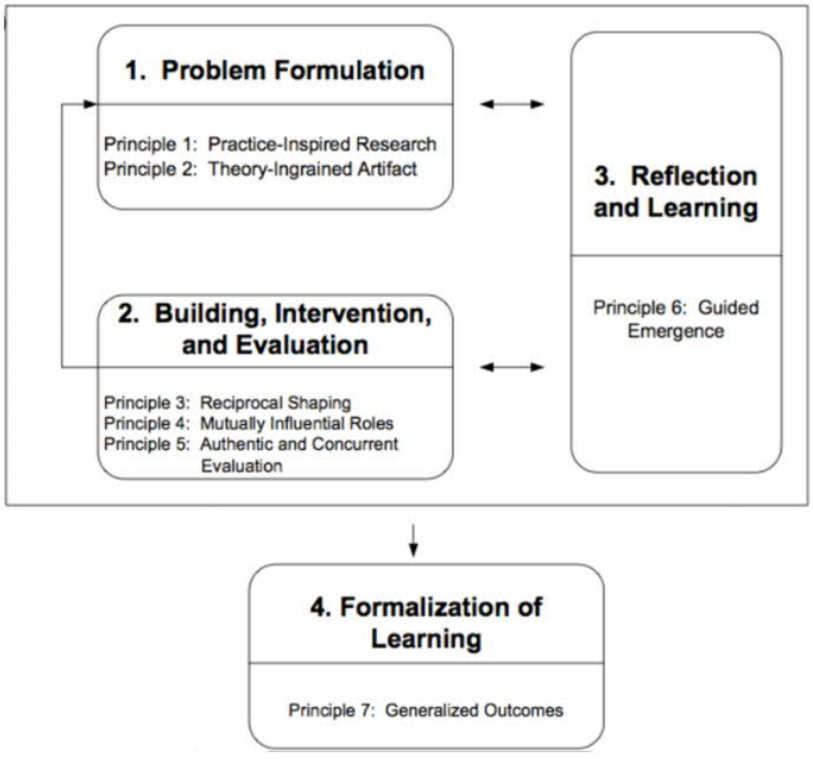

Figure 2. The ADR method: Stages and principles (Sein et al., 2011).

3.3 Trustworthiness of the study

To ensure the trustworthiness of the study, all the stages and principles of the ADR were closely pursued. The clear guidance and steps in ADR make it easy to follow also from the empirical viewpoint (Rogerson & Scott, 2014). The ADR method supported the solving of a real-life problem that had emerged in an organizational setting. The organization found the research valuable considering their future projects and arranged all the necessary material and access rights for the researchers enabling a thorough study on the matter. In addition to the full time of the researchers, the project utilized the time and effort of several system-, process- and RPA experts that formed the practitioner and end-user groups. The result, stemming from generalized learnings that were enabled by several participants, project cycles and iteration rounds, was quickly taken into use as the automation solution was scheduled to operate on its own partially already during the final testing phase. Hence, the trustworthiness of the study was built up by wide participation of different experts with dense iteration periods.

4 Empirical study

4.1 Collection of data

There are three different participating groups in ADR that collaborate to the result. They are researchers, practitioners, and end-users (Sein et al., 2011). This research project included two researchers who work as external RPA consultants for the organization. The researchers were also responsible for practical implementation of the automation solution. Four members of the organization's RPA team formed the practitioner group and the end-user group consisted of the SMEs, different system experts and other type of potential end users of the process automations being designed. Empirical data in this study was collected throughout the project in meetings with workers from the case organization and a third-party organization who gave input on the artifact under production based on their areas of expertise. The meetings were held mainly in Skype with a couple of exceptions when a face-to-face meeting was possible.

The first phase of the study consisted of gathering data on the tasks being automated. First, the SMEs, in other words, manual process experts for the employee data management were identified and contacted. The SMEs at this point included one employee from the target organization and one employee from a third-party organization. The data was gathered in several workshops and meetings between the researchers, two members from the organization's RPA team and the SMEs. The SME informants chosen in this phase were one process owner in the case organization's IT administration and one group manager in the third-party organization's IT department. During the workshops there was also a need to arrange meetings with system experts who had more insights on the logic of the applications used in the process to be automated. The system expert informants were members of the case organization's IT administration department.

Having gathered data from the first workshops and meetings, follow-up meetings where held between the researchers, SMEs, and an external RPA consultant where we first decided to divide the structure of employee data management into seven separate processes as there were seven different modifications that could be separately ordered for any employee's user accounts. Each of these modifications had their own form to be ordered with which created a service request ticket to one of the applications in use. Because all the

processes were triggered by the service request tickets in one application, we then decided to add another process into the solution that only screened tickets in that one application, made certain validations to them, and issued them to the other seven processes based on which kind of request was ordered. The process also included screening a comment section of tickets, which was a part of almost all the request types. In addition, the starting and terminating the applications used in the solution was decided to perform via a standardized application controlling processes. During this stage, eight process definition documents (PDDs) were also created to describe the steps included in each process in detail.

After the processes had been defined and documented, and the initial development of the automation solution had begun, we arranged meetings between the researchers, the SMEs and system experts. The agenda in these follow-up meetings was to present and review the initial design of the automation solution. The meetings produced valuable improvement ideas as all the participants viewed the project as an important benchmark for the future RPA implementations in the organization and saw its potential in significantly speeding up the processes of employee data management.

With the feedback gathered, we started to improve the solution. It was noticed that the RPA solution would allow some parts of the processes to be handled differently than in the manual version of the processes handled by human. In other words, there were shortcuts to be taken as a result of transferring the execution of the processes from humans to robots. The biggest shortcuts involved three old user interfaces, or tools, in three of the processes built several years before to help the employees make the requested changes for an employee account located in an active directory (AD). The data in AD could be edited by sending Windows Powershell commands or by using the AD interface, both of which took a significant amount of time for an employee to execute. The tools provided an easier option. With the RPA solution, these three supportive tools could now be bypassed as there was already a component integrated in the RPA software that could execute Powershell commands even without opening the Powershell's own interface. In other words, all the processes could use the same Powershell component to manage any modifications or queries in AD by just adding the commands to the RPA solution.

We also had meetings with the product owner and head of the organization's RPA team who had been developing the first automation solution of the organization. In those

meetings we searched for touchpoints between the first automated process and the eight that were now in development. These touchpoints could include applications used in both solutions of otherwise similar parts of automation logic that could be re-used as such or with little modification both in the solution in question and various other processes potentially developed in the future. An improvement measure at this stage was to modify and split some previously created RPA components into smaller components to enhance their reusability to several different processes.

After the improvements were made, the testing of the solution was accepted by the business unit in the organization to be moved from test environment to production environment meaning that the robot would now start to handle real cases instead of mock cases. At this stage, several testing sessions were held. The participants in these sessions were one researcher, members from the organization's RPA team, two SMEs from the third-party organization and one systems specialist in the case organization. The systems specialist had comprehensive knowledge of both the applications and the business processes being automated and coordinated the final testing phase. The final improvement ideas that arose in this stage included creating a master process, another hierarchical process layer, to control all the eight processes. The master process was designed to send customized orders for the other processes when to start executing and when to stop based on for example the current time and the if there were any cases to handle. This helped manage the running schedules of the processes in the solution. Another improvement at this point was that part the process screening the tickets and comments in one of the applications was modified and made into a separate standard component to the component library to be able to serve other potential future processes that might require screening of any kind of tickets. This was seen as an important step as the ticket system had a central role in the organization's operations. The last major change in the solution was adding one more process to it that created a daily report of the statuses of all the ticket cases that had been handled by the robots.

Around 50 meetings were held during the whole project with a variety of stakeholders, assessing several different aspects. Comprehensive notes were gathered in all the meetings. A summary of the data collection process is presented in Table 1 below. It includes the cycles in the second stage of ADR study that is called Building, Intervention and Evaluation (BIE) (Sein et al., 2011), and the key events with the data collected in them.

Table 1: Summary of data collection process.

Time period and intervention cycle (if applicable)	**Event**	**Data gathered**
June 2019 – July 2019	Meetings with SMEs and system experts	Mapping of the tasks in employee data management.
August 2019 (first BIE cycle)	Meetings with an RPA expert	The decision to divide the tasks into eight automated processes and create standard components for controlling applications.
September 2019 – January 2020 (second BIE cycle)	Meetings with the head of the RPA team, system experts and SMEs	The decision to bypass three old tools by replacing them with a single component executing different Powershell commands in the automation solutions. Another decision to divide some previously created RPA components into smaller components to make them more reusable.
February 2020 – April 2020 (third BIE cycle)	Meetings with SMEs and members of the organization's RPA team	The decision to create a master process to control all the processes in the solution and enhance the reusability of the ticket system section. Another decision to add a reporting process in the solution.
May 2020	Scheduling of the RPA solution to fully run on its own	Acceptance of the artifact.

4.2 Problem formulation

Evoking the idea for a research was the situation in the target organization concerning RPA. The organization was at the beginning of their RPA journey and needed help to get things moving to the right direction. Like many other large organizations, they had acknowledged a great potential to reduce costs, minimize errors and speed up processes with RPA as the nature of work in such organization included endless amount of repetitive and rules-based tasks that could be automated. The plan was to start with a decent pace by using both external RPA experts and beginner level in-house RPA developers in the first projects and then gradually develop their own RPA capability towards running RPA as much in-house as possible. In other words, they not only wanted to implement RPA as efficiently as possible from the start but also invested heavily in training their own employees to form an RPA expert team. Also, to maintain decent funding for the RPA

department, the first implementations needed to show fast and calculable benefits as is the promise of RPA. This further amplified the pressure maximizing the quality of the first implementations and building a solid fundament for future automations. It was also clear that the first automated processes needed to be the most potential ones in terms of showing the quick savings and other benefits.

The RPA team in the organization was divided into two separate units. One unit searched for candidate processes for automation by gathering ideas from the employees, evaluating the potential benefits and obstacles of applying RPA to them. This unit was also responsible for mapping all the steps of a process that was deemed suitable for RPA and creating a PDD document that included every detail needed in developing the RPA solution for the process. The other unit focused on developing the RPA solutions for the processes based on the PDD documents.

The first unit had screened a process managing the user accounts of the employees in the organization that had a great automation potential and seemed suitable with the situation. It was decided that the researchers take responsibility of mapping the steps of the process with the help of SMEs and create the PDD. After that the researchers would also be responsible for developing the RPA solution. At this point, the researchers discovered that the process of managing the employees' user accounts was relatively large and would possibly be divided into several processes. Also, the process interfered with two of the most central applications used in the organization and AD that contained all employee data. This combined with the fact that the organization was planning on implementing countless other processes in the future that would most likely be using the same applications and systems introduced the challenge of creating automation components that could not only be scaled and reused within the RPA solution in question but also in other several automations just by managing the inputs and outputs of the components.

The project was seen as an important benchmark on how the principles of modularity worked with RPA in different scenarios. This required studying the factors affecting the degree of modularity in different situations. In other words, the effects of modularity would be assessed and adapted to the research setting in this context. Thus, finding the way to increase modularity in the components here would not only enhance the effectiveness of

this implementation but also future implementations in the organization through providing reusable components and educational value.

4.3 Building, intervention, and evaluation

Once the problem was formulated, the ADR team proceeded towards the first version of the automation solution and engaged in the BIE cycles of ADR study presented by Sein et al. (2011). The purpose in this phase is to build, review, and improve the solution in intervention cycles involving all the teams of the ADR study. This research ran through three intervention cycles as shown in Figure 3 below. The first cycle aimed at breaking the structure of employee data management into separate automation processes and creating detailed PDD documents for each process describing the tasks included in them in great detail. The first cycle also included sketching the initial design for the solution. The goal in the second cycle was to design and develop the automation solution in the organization's testing environment based on the PDD document data and improvement ideas in the first cycle. In the third cycle the RPA solution was tested and validated in the organization's production environment by handling real cases and final modifications were made.

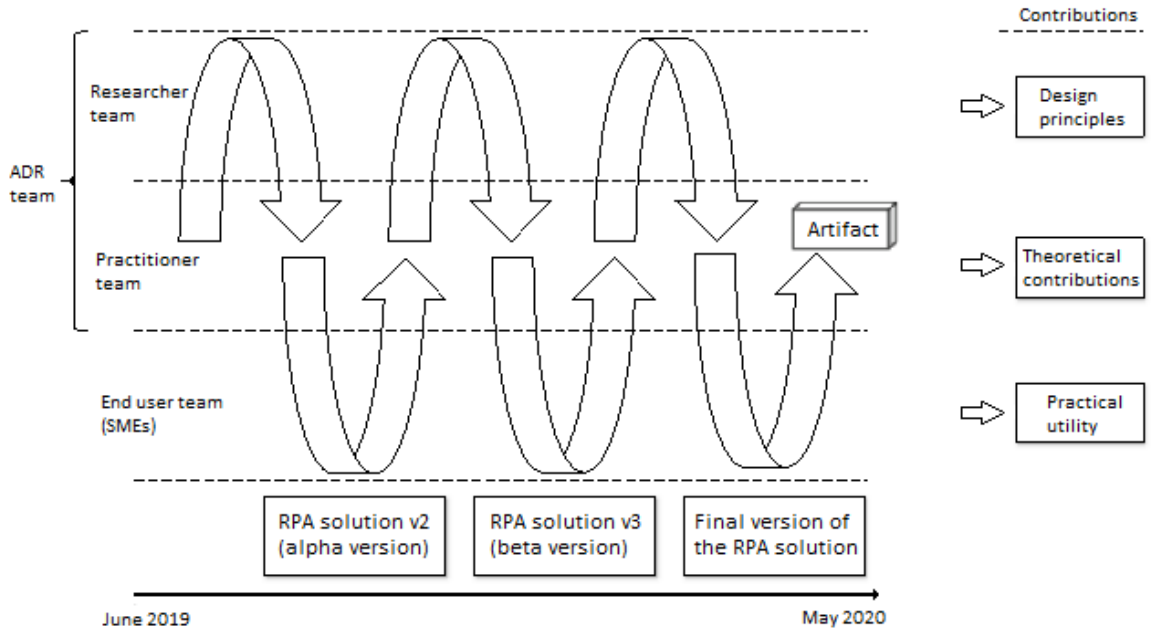

Figure 3. Intervention cycles for the artefact for The Finnish Tax Administration.

The goal was to produce an artifact that in practice would execute all the requests made for user accounts of the employees in the case organization. The artifact had another, perhaps more important requirement as well. Because the organization was in the beginning of the

RPA journey and aimed for a steady start with RPA utilization, the artifact needed to be versatile and work as a vanguard RPA solution in the organization in terms of how modularity was applied in RPA components and what benefits it had.

The RPA technology used in the project was based on "object-oriented" robots whose tasks can be stored and reused in an automation component library. An RPA component library allows each task only to be defined once and then being able to be pulled from the library and applied to as many different automations as needed. The value of this component reusability can be exponential. The more processes are automated, the more components are built in the library. Thus, the more re-use conducted, the more economics can be achieved in assembling and delivering those components into new processes. The components in the bottom layer of the RPA component hierarchy in the RPA software used in the project are called "objects". The objects are called by the "processes" to usually perform a simple function. The processes, that can also be called RPA components, typically consist of several other RPA components. These components can be objects or other processes as subprocesses. The subprocesses, in turn, may consist of several other components. Thus, an RPA solution forms a hierarchical component structure where different commands or conditions can be forwarded downwards through inputs of components. Objects usually just include a simple function such as writing something to a text field or clicking a button in a page. Every RPA component, whether located in process layer or object layer in a solution, has a certain mission. The components interact with each other by the rules configured in each process and the specifications of their inputs and outputs. The processes are executed by runtime resources, robots. When a process is run, the robot executes one process step, often an object or subprocess, at a time until all the steps in the chosen process path have been gone through. If there is a need to speed up the case handling in a process, more robots can be set to execute it simultaneously. However, the number of robots is limited by the license purchased. There might be several process layers in a solution, depending on the business process being automated, but usually only one object layer. An illustration of the process structure created in this project will be presented in the next chapters.

4.3.1 Mapping the manual process execution

Before starting the designing of the automation, the employee data management process had to be mapped from a manual process execution perspective. This meant going through

the whole process with the designated SMEs. The process starts with screening service request tickets related to the employees' user accounts in the ticket application called "Piste". One ticket here equals one case and the cases are handled one at a time. After picking the first ticket from the ticket system, the following measures depend on what kind of request has been made with the ticket. The requests include adding a new user account, removing a user account, activating a user account, passivating a user account, changing the name of the user account, changing the expiry date of a user account, and changing the organizational department of a user account. All the measures, however, require editing information in AD that can be done with a separate tool or directly in AD interface depending on the request type. Information in another application, going by the name "Gentax", that also stores some employee information is edited in all of the cases as well. After that, the manual process executor goes back to the Piste application to create notices to different parties in and outside the organization about the modifications that have been performed. Some of the request types have similar policies concerning the notices that are made but there are variations. The notices are followed by marking the ticket as solved or partially solved in Piste and moving on to handle the next ticket. For a partially solved ticket, it is required to make a check later whether a third-party organization has sent a comment to the ticket in Piste, confirming that measurements on the case have been completed on their side. Only after that can the case be marked fully completed. The abstract of the initial employee data management process mapping is seen in Figure 4 below.

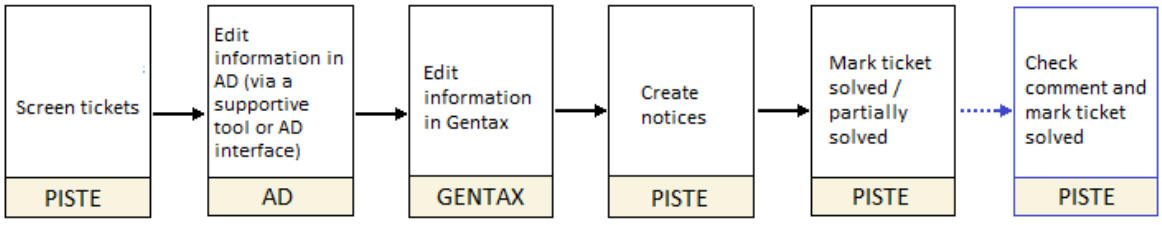

Figure 4. The employee data management process from manual viewpoint.

Towards the end of the process mapping workshops, an additional meeting was held between the researchers and the SMEs to look at the process from automation viewpoint. The large size and complexity of the mapped process produced challenges from both development and control viewpoint. For example, the management of the different service level agreements (SLAs) that the request types had was clumsy with one automation

process. Some of request types also had a significantly larger number of cases per week than the others and their automation was thus potentially more valuable. The result of the meeting was that instead of one process it would be beneficial to divide the employee data management into seven automated processes based on the seven different requests that could be made for the user accounts. This division made sense because the requests were always made in separate tickets and handled independently. Having all the request types and their measurements handled in one automated process would make it very complex as the process would have a massive number of different paths and steps included in it.

The division enhanced the controlling of the automation as the execution times and the number of robots executing different requests could now be independently set for each one. Thus, the different SLAs that the request types had and the unevenness in the numbers of different requests at different times could be managed better. If there were for example excessive amounts of certain kinds of tickets, the process handling those kinds of tickets could be assigned more robots or set longer running times to balance the situation. Furthermore, if there were enough licenses, the processes could now be simultaneously run to handle the user account requests faster as a whole. Now it was also more convenient to develop the solution because the developers could each start designing their chosen independent part, a process, in the solution concurrently, instead of trying to coordinate the development of one and same process. Another benefit was that the solution could now be gradually put into action, generating the RPA benefits faster, as each process could be independently finished and scheduled to run, the most valuable ones first. The potential bugs and other general attributes concerning the automation of the applications could also be exposed when testing the first automated processes which could then be taken into account in the development of the other processes in the solution, thus smoothening the development project. One more benefit of this division was that the effects of potential errors emerging in the execution of the automation were now isolated into one of the seven parts. A process in error state could be fixed while other six still ran normally. With the decisions made at this point, the initial automation flow of the RPA solution was designed (see Figure 5).

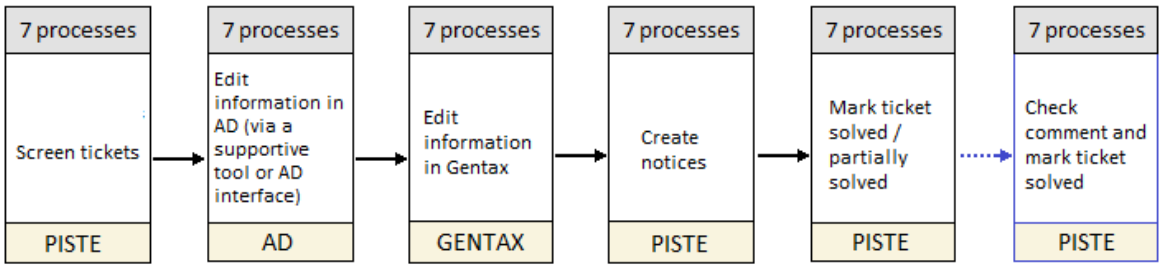

Figure 5. The employee data management process for automation (RPA solution v. 1).

4.3.2 The first BIE cycle

After breaking the process apart into seven processes another meeting was held between the researchers and an RPA expert to review the initial design of the automation and produce potential ideas on how the solution could be made even more controllable and/or resilient. Basing on the ideas in that meeting it was decided that the solution would be divided further into total of eight processes. Because all the inputs for executing the user account modifications came from the tickets in Piste, we saw it beneficial to dedicate one process solely for the ticket handling. The purpose of this process was to screen the tickets related to employee data management, make quick request-related validations, categorize them, and distribute them to the other processes based on the request types. Another purpose for it was to check the comments from the third-party organization mentioned before and mark those cases fully completed. This additional process had multiple benefits. Firstly, unnecessary identical copies of the component structure handling the ticket screening and comment checking parts in the seven processes were removed as the object components interacting in those parts were now centralized into one process component that would be executed in the beginning of each day. This made the automation of that part faster to edit as only one process needed accessing instead of seven. Secondly, the controlling of the solution was further enhanced as the ticket screening and comment checking could now be scheduled to run independently and executed by separate robots, as many, as the situation needed. Also, if the ticket screening process would not find certain types of requests in its execution, the processes handling those request types would not have to be needlessly started that day. Gathering all the tickets for the processes simultaneously instead of each process opening the ticket system separately and searching for certain tickets also minimized the robot's navigation actions in the ticket application, speeding up the process and reducing the risk of potential system related errors. The filtering options in the ticket system also allowed showing all commented tickets listed in one view. This speeded up the comment screening part of the process and reduced needless

navigation in Piste even more because the robot did not have to search for each previously handled ticket separately in Piste to check if it has been commented. Instead, it could just directly view all the tickets that had been commented and mark them fully solved.

Like the ticket and comment screening process, both the launching and terminating of the two applications used in the solution were also separated into their own processes. The difference to the ticket process was that the application controlling processes were placed lower in hierarchy in the solution as they were placed as subprocesses in the other processes. They could be given orders as inputs by the other processes on whether to start, restart, or terminate, depending on the situation. Another difference was that the application controlling processes could also serve any other processes outside the current solution that were using the applications in question, while the ticket screening process was configured in a more customized manner for the use of the current solution. The applications were central to the organization so having standard components for controlling them made perfect sense. Instead of wasting time figuring out and battling with the extra functionalities related to starting or terminating the applications, the developers could now just pick the ready-made application control process bricks from the repository and place them in their solutions. The only thing needed to do was setting the control order as an input in each scenario where the component was used. Stemming from these decisions, the second version of the RPA solution was then designed. It is shown in Figure 6 below.

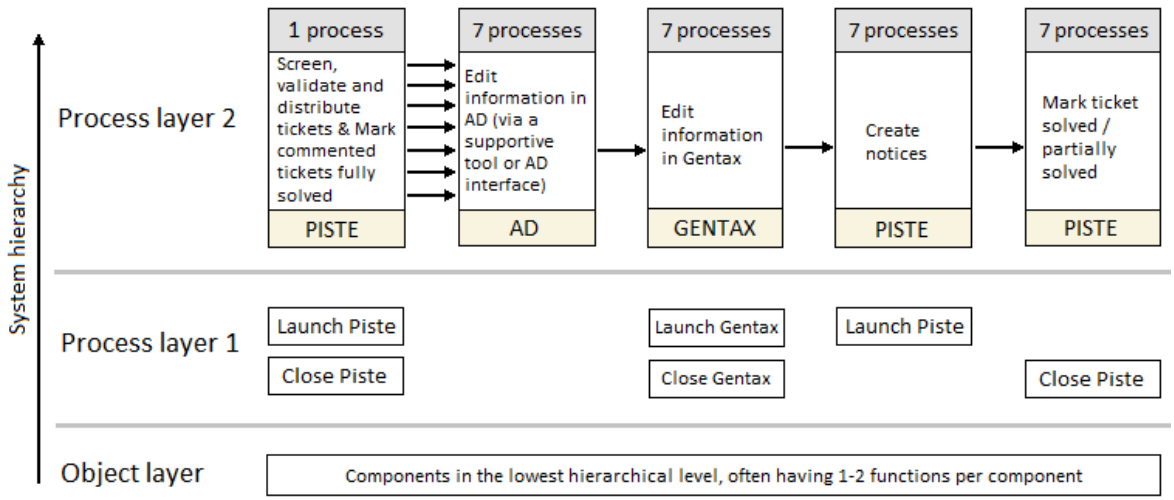

Figure 6. The employee data management from automation viewpoint (Solution v. 2).

4.3.3 The second BIE cycle

The second version of the solution was gone through in several meetings with the head of the RPA team, system experts and SMEs. There were two significant improvements ideas presented at this stage. The first one was related to old tools included in three of the processes. In three of the processes a tool was used to perform the requested changes in AD. One tool was for adding a new user, another was for removing a user, and the third was for editing names in the user account. These tools, launching Windows Powershell commands, were initially made to help the employees make the changes in AD and speed up their work. In practice, the information from the service request ticket was filled to the fields of a supportive tool and by pressing an ok button the tool executed Powershell command to make the requested changes in AD. The RPA software had shown some difficulties attaching to and interfering with the elements of the tools which brought up concerns about the increased risk for system related errors. At the same time, the implementation team was battling with the high number of other AD-related tasks in the processes that required navigating in different parts of AD user interface. An idea was proposed that both the tools and AD navigation could be bypassed because robots would not have any problem directly opening the Powershell interface and executing the corresponding Powershell commands with just varying the parameters for each situation. Furthermore, an RPA object component was discovered that could execute Powershell commands directly in the RPA software where it was integrated. As, in fact, all the seven processes handling the requests were making modifications and/or queries in the Powershell-commanded AD, they could all utilize that same component by just feeding it different commands as inputs. This improvement speeded up the execution and development times of both the processes in the current solution and potential future solutions by saving a lot of unnecessary clicking and field-filling from the robots. Automating the editing of AD information was now quicker in any scenario because any detail in AD was modifiable through one component, instead of configuring the robot to navigate in different parts of AD or in the supportive tools for each individual modification. The Powershell component, or object, also mitigated the complexity in the automation processes by replacing the instructions for robots concerning clicking buttons and filling fields with direct commands enabled by coding that was hidden inside the component.

The second discovery in the meetings at this stage was that there were some objects used in the processes that included almost identical functionalities. An example of this was two object components configured to perform functions in the same page of the Piste application when editing the status of a ticket. The first one was configured to empty one field and fill three fields with values given as inputs to the component while the second one was otherwise identical but filled only two of those fields. Both the components also had a function of pressing an ok button after all the other functions were performed. The components were used in different scenarios among the first automated process and the eight under development. The discovery was made because in one of the processes under development a need emerged to perform a different combination of actions than what the two already made components offered. When discussing about the page's commonality in the organization's business processes with the SMEs, it was concluded that there would most likely be other occasions in the future as well where different sets of functions needed to be performed in the page. Therefore, the automation solution for the page needed to be more versatile, enabling all the different combinations of functions to be executed.

Because the page was designed to allow any number of fields to be filled with any values independent of one another, the functions in it were designed to be very weakly connected. The configurations in two object components in the RPA software, however, linked the functions strongly together, which created an imbalance. A decision was made to break apart the components and create one component per one functionality in the page. This transformed the initial components, whose usage was tied to certain business process scenarios, to several components that could be used independently in any process and any scenario that interfered with that page. The components could also now be mixed and matched freely, whatever was the number of fields needed to be filled in that page. Another modification was to transform the component that emptied a field into a similar component with the others, meaning that instead of always emptying the field it would be given the value to fill to the field as an input. This increased the number of variations that could be made with the automation components performing the functions and matched that number with the number of functional variations provided by the page's design. In other words, the relations between the functions were now at the same level from the application's viewpoint and the viewpoint of the automation software. This modification also erased the needless overlapping of functionalities in the initial two components. With these improvements in mind, version 3 of the solution was then developed (see Figure 6).

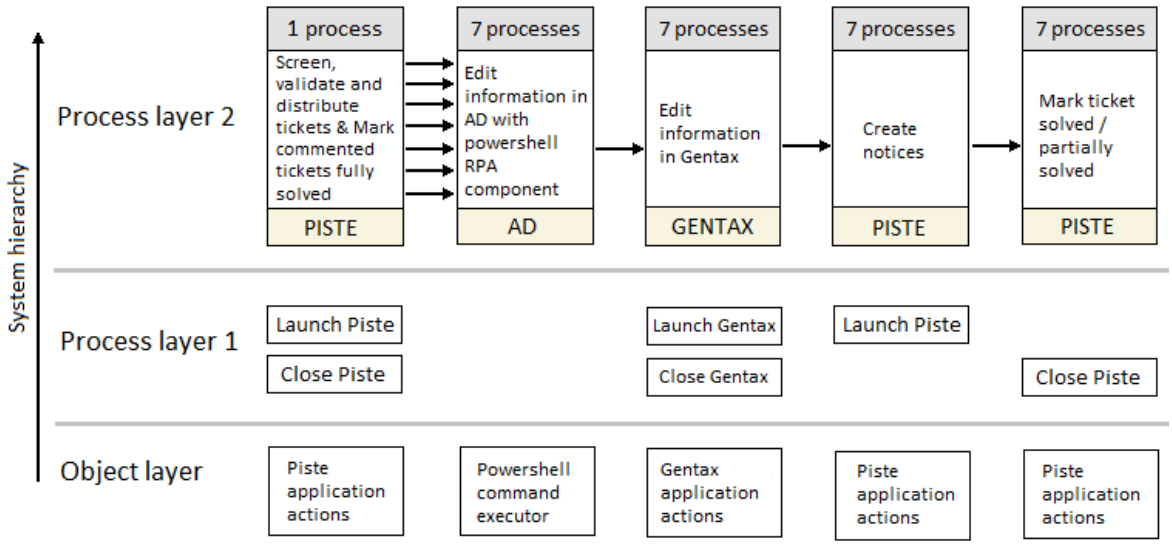

Figure 7. The employee data management from automation viewpoint (Solution v. 3).

4.3.4 The third BIE cycle

After creating the third version, the RPA solution was moved for further validation in the organization's production environment meaning that it would now handle real cases while testing under surveillance. This production validation period took three months and generated valuable improvements. The involved parties were the researchers, the SMEs, system experts, members of the organization's RPA team and one consulting RPA expert. After this stage, the solution was considered ready and scheduled to run on its own in the production environment.

There were three major modifications performed to the solution based on the ideas that were put forward in the meetings and testing sessions during the final validation period. The first idea was to isolate a section of the ticket and comment screening process into a process of its own to be placed in the component library, and thus enhance its reusability and scalability more widely in the organization's RPA projects. This meant creating a standard process component that could be used in any other solution screening tickets as well. In practice, the ticket screening in this solution began with opening the Piste application and navigating to the ticket list. This was followed by opening the filter property next to the list, setting values to the chosen filters, and clicking the search button which updates the list to match the chosen details. The robot would then read the whole list of ticket numbers into its memory and start accessing the tickets one by one filling a ticket number from its memory to a general ticket search field in the application.

The sequences of actions starting from opening the ticket list and ending with reading the ticket numbers into the memory of the robot was considered a very common phase in the beginning of business processes in the organization. In other words, several automations in the future would most likely be configured to go through that same procedure. Thus, we decided to isolate the object components performing these actions into an additional process with standard input and output specifications and publish it to the RPA component repository. This process could then be spotted and retrieved from the repository to be used in other solutions that needed to filter certain tickets from the system. Before publishing the new process, a modification was needed to enhance its functionality and suitability for different purposes. Instead of only including the filters needed in the employee data management solution, all the rest of the filtering options provided by the page's filter tab were added into the functionality features of the process. In other words, more objects were created and added to the process, each object handling one filter. After adding the objects, the new process had the same standard amount of filtering options that could be fed as inputs, as the filter tab in the ticket application had, and it would return the list of ticket numbers as outputs in a solution for further processing. The automation options were once again matched with the options provided by the target application's page. Having a separate component for the ticket system had similar benefits to the standard application control components. It streamlined the usage of the ticket system section to processes outside of the current solution, speeding up the future RPA development projects that would include ticket system actions. It also simplified the structure of the main process layer in the current solution as a new component brick in the form of a subprocess was introduced to replace a set of object component bricks, hiding a significant amount of information in, formed by the newly replaced ticket system objects and their relations.

The second major improvement in the solution added a new level of process hierarchy into it. It was noticed editing the running schedules and regulations for all the processes in the solution was a clumsy and time-consuming activity. There were relatively large variations in case amounts, run schedule requirements, and SLAs among the processes. Furthermore, the situation was changing constantly. The frequent need for modifications to the execution settings of each process called for enhanced convenience in managing the settings. To manage the solution faster as a whole, a master process was created. The master process was an additional process layer on top of the other processes in the solution. All the processes could now be given instructions on for example when to start running and when

to end via inputs to the master process. The master process could simply be executed itself, passing on the customized instructions for each process under it. This saved a lot of time when editing the schedules or other running conditions for the eight processes, because all these details could now be set simultaneously through one process instead of eight. It also helped manage the complexity of the solution in the sense that the whole structure of the solution could now be viewed and operated in one process page, each process hiding the information related to its execution.

The final major modification to the solution was adding process to the main process layer that created a daily Excel report including details of the handling of the different cases from an ongoing day. These details included for example the number of completed cases, the number of incomplete cases resulted from system errors, and the number of cases transferred to manual handling because of insufficient information in the service request ticket. The reporting was first planned to be the final part of each seven processes separately but creating a mutual report after the execution of all the processes in a day saved time from the robots as only one Excel file needed to be created and filled once a day. The adding of the reporting process in the structure was followed by a conclusion that the RPA implementation for the employee data management was completed and the artefact was accepted. The fourth and thus far final version of the solution can be seen in Figure 8 below.

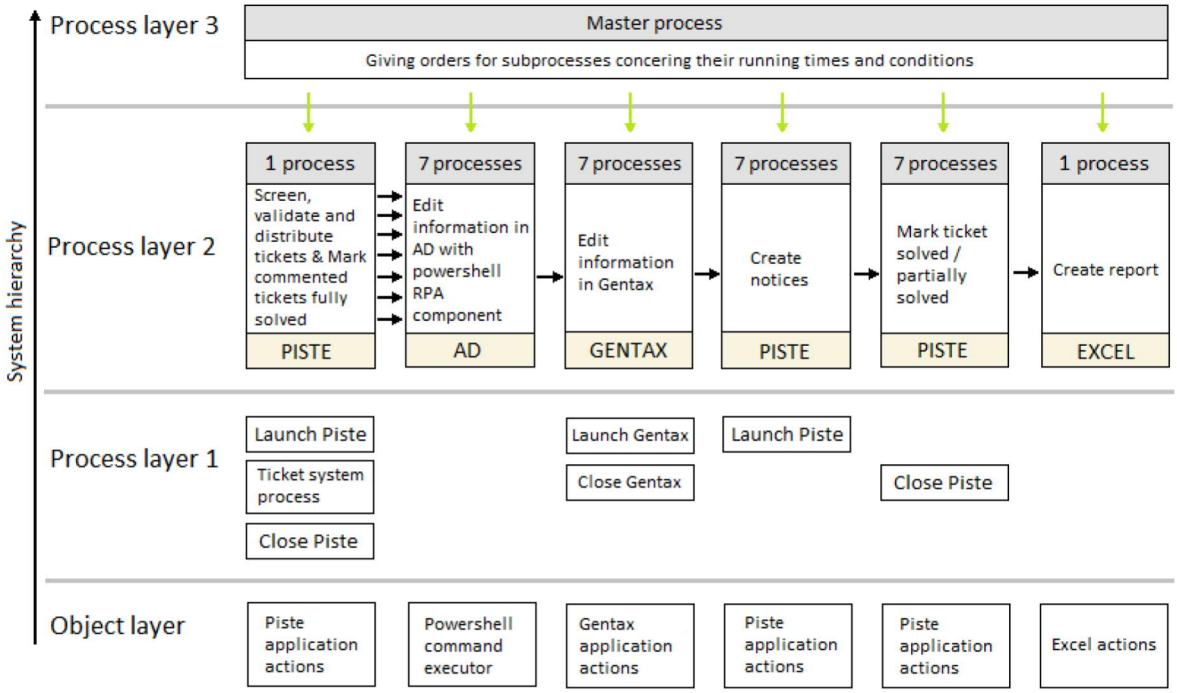

Figure 8. The employee data management from automation viewpoint (Solution v. 4).

4.4 Reflection and Learning

Having the solution performing as intended in the organization's environment, the implementation could be called successful. However, throughout project there were situations and outcomes to learn from. The reflections on those situations in this chapter are a result of discussions had and notes taken during the project.

The first thing to learn from was the structure of the project in terms of how the processes were developed and put in action in the production environment. Dividing the initial employee data management process into seven separate processes based on the request types in the beginning enabled having part of the solution generating the savings and other benefits in a relatively quick schedule. Compared to battling with complexities of one massive process for a long period of time, the seven processes could now be developed, validated, and put into work gradually. For example, the process handling user account removals could have been finished and scheduled to run on its own while the still designing the rest of the processes in the testing environment. In other words, decomposing the employee data management structure from automation viewpoint enabled the implementation project to be similarly decomposed. In fact, getting the full benefits out of the division to seven processes would have required the schedule of project to be similarly

decomposed to seven independent parts. This advantage was not fully utilized however, because even though the processes were somewhat gradually set to work in the production environment, they were almost all developed and put through acceptance tests in the testing environment before the first one was even moved into production environment validation.

The second learning point comes from centralizing some of the ticket application components by creating the additional ticket application process to screen the tickets, categorize them, and distribute them to the other processes for further handling. This process also had another part that checked the comments from a third-party organization and marked those cases fully completed. The ticket system section was later molded into another separate RPA process with standardized inputs and outputs because it was noted that it plays a central role in several business processes that the organization would be potentially automating later. The new process was configured to be combined with any processes needing to screen tickets. Similarly, the launch and termination of the two applications were separated into their own processes having configurations that enabled them to be combined with any processes that used those applications. All these processes were placed in the component storage to be spotted and retrieved. The modification and publishing of the ticket system process was conducted at a later stage in the project and there were already a few other automation projects underway at that point. Those projects might have benefited from such component. Thus, it was concluded that the search and assessment of common tasks, would be most valuable when done already in the business process mapping stage with the SMEs. The processes automating those parts should also be created first in the development stage and published into the repository as fast as possible to generate the scaling benefits of component development. This would, of course, be preceded by checking that there was not already a component for the same purpose.

The third aspect of learning is closely related to the second one. It refers to realizing the possibility to bypass the old tools created to help update the organization's AD. After battling with the poor co-operation between the RPA software and tools for a while, the team discovered that using the tools could easily be bypassed as they were only supportive "interfaces" made to use the information filled in them as parameters in Powershell commands that edited information in AD. There were also some parts in the processes that needed to retrieve info from AD that had not yet been developed in the solution. At this

point, an RPA component that could execute any Powershell commands given as inputs was discovered and brought into the component storage. All AD related queries and modifications in any process could now be performed with one component, saving a lot of extra work from developers.

The fourth learning came from discovering the two almost identical components that included a set of functionalities performed in a single page of the ticket application. Since from the application viewpoint the page was designed so that any number of fields could be filled and then saved, it was not efficient to have multiple object components with different sets of functions performed in the page, tightly coupled with different scenarios. Thus, each of the different functions that could be performed in the page was automated with their own object component that was named after the function. Now the developers could use a desired amount of the page's functionalities in their automations just by picking the matching components from the storage, instead of examining the sequence of multiple functions in a variety of components made for different scenarios in the page. Thus, assessing the hidden information in the components could now be skipped.

The final major learning came with the clumsiness of controlling the running times, running conditions and recovery logic for the processes in the solution. Having them all separately accessed and regulated took a relatively long time. Towards the end of the implementation, a control process was created to solve this problem. This meant practically creating a new process layer on top of all the eight processes and their subprocesses. All the controlling could now be conducted through this master process that would pass on the running instructions for the subprocesses given as inputs. In addition to saving time, the new hierarchy level provided by the master process also helped the robot administrators to visualize the solution and its logic as a whole better, and thus manage the complexity of it.

5 Discussion

5.1 Formalization of Learning

5.1.1 Design principles

This study formed four design principles that steer RPA solutions towards higher modularity. The first one is about decomposing the whole solution into several processes, the second one is related to decomposing the components into smaller components, the third one covers creating separate components for the common parts of the processes, and the fourth principle handles adding a new process layer to an automation solution.

Design principle 1: Decompose a process for automation into as many processes as there are different types of independently executable cases originating from its source(s) of information

The first design principle is related to detecting the different independently executable scenarios that the entity being automated includes. The scenarios here mean different types of cases, each having their own sequence of steps that significantly differentiates from the ones in other types of cases. If these cases can be independently handled, and thus the automations for them can independently produce benefits, the modularity level of the solution should be increased by decomposing the solution into as many automated processes as there are case types. This allows a faster reaping of the benefits from the automation as the solution can be gradually deployed. The automation for handling the most valuable case types can be created first and put into action, generating cost savings and other benefits already before the other parts are developed. Majority of the bugs or special features related to automating the applications are also exposed when test running the first automated processes. These properties can then be taken into account when developing the rest of the processes, saving time from the developers. A faster implementation process is enabled also because different parts of the solution can be independently and concurrently developed. Another benefit is limiting the effects of potential errors in the solution into a smaller area as they only impact the scenario they emerge in.

In this project, the solution was initially split into seven automation processes in correspondence with the seven different types of cases based the seven different types of service requests made for the user accounts of the employees. All the requests originated from one source, the ticket system in the ticket application, and were handled by different sequences of tasks. The splitting made it easier for the two developers to start developing the RPA solution concurrently as they could each work on different independent parts of the employee data management without excessive need for coordination between each other. It also enabled automating the handling of the most common and valuable requests first and launching their automations before other parts. Another benefit gained was the enhanced controlling of the solution as there were now seven parts in the solution to be separately regulated in terms of running conditions and the number of robots, or "resources", executing it. This helped for example keeping up with the different SLAs that the requests had by adjusting the conditions and designating more resources to the processes that handled the most urgent request types. One more advantage gained was that the potential errors in the execution of the solution were now only affecting the handling of the request type they emerged in.

Design principle 2: Increase the number of different automation configuration options by decomposing automation components into smaller components with less functions whenever it enhances their reusability in different business process automations

The second design principle has similar traits with the first one but focuses on the automation components in the lower hierarchical levels. It highlights the meaning of ties between automation components and business processes. The components, usually operating at the lowest level in the hierarchy, should be decomposed to a point where another decomposition would not enhance the reusability of that part of automation in other automations. In other words, the variety of possible automation configurations for a certain area should match the variety of the needs of potentially automated business processes operating in that area. This steers the automations towards increasing flexibility and efficiency by creating sets of components, each set containing one type of function in one spot instead of several, that offer at least the same amount of automation configuration options for a certain area that the business processes use functionalities in that area. It often means matching the number of different automation options with the number of functional options provided by the target applications or their parts by creating automation

components that each have a limited number of functions to perform, typically one or two per component. The number functions in a component in this case depends on the relations of the functions in the target application. This procedure allows all the different business processes using those parts to be efficiently automated combining the same standard components for each purpose, instead of making the developers browse through a variety of components with different combinations of functions in each scenario, and trying to find out if there is already a component made for their purpose.

In this project, the ties of RPA components to certain scenarios and business processes were loosened while examining the functions in Piste. The two almost identical object components with several functions were transformed into several object components, each having one function related to the page they were interfering with. Because the functions in the page were once designed to be used independently or with any number of other functions by the software developers, this modification enhanced the reusability of the automation made for that page as a whole by allowing the free mixing and matching of the functions in the page via their own RPA components, thus adjusting the level of relations between the functions from automation viewpoint to match the level of relations provided by the target application. In other words, the two initial components that were tied to certain business process scenarios where a certain set of functions in the page was to be performed, were now split into several different components consisting of one function each that could be mixed and matched for any kind of purpose that the page design allowed. This modification was necessary as it was discovered that the page in Piste was frequently used in a variety of business process scenarios, both the ones being currently automated and other processes potentially automated later. This meant that that all kinds of combinations of functions were used in the page and should be made possible to automate as well. If the page, on the other hand, would have been in rare use in the business processes and/or there would for example most likely be only one combination of functions used in the future as well, similar pressure to create multiple components and diversify the automation options would not have existed. Identical modifications, however, were done to some other parts of the solution as well, producing the same benefits.

Design principle 3: Create separate automation components for the common parts of business processes being automated

The third design principle focuses on the automation of the common tasks shared by different business processes and spread across the automation processes. An automation design can be improved by detecting these features and grouping them into a single automation component. This speeds up the development work by enhancing the economies of scale as those components handling the common parts can now be easily reused wherever those parts are interfered with, instead of constructing the same sequence of steps repeatedly in different automations.

In this project, the third principle was followed on several occasions. Creating an additional eighth process for the RPA solution containing the common parts of the ticket application shortened the development time when all the modifications concerning those parts could now be made editing one process instead of all seven processes. The risk on system errors was also mitigated as the robot would only have to navigate in the ticket system to screen tickets and check comments for all the request types on one occasion instead of a total of fourteen times during a full execution of the solution. This also further added control to the solution as the screening of the tickets could now be separately regulated in terms of running conditions and the number of robots executing it. Similar advantages were gained by for example creating one report process for the whole solution, isolating the launch and termination functionalities of the applications into their own processes, and centralizing the editing of AD into one component that executes Powershell commands.

Separating the sequence of steps in the ticket system further into its own process was also an application of the third design principle. The difference to the creation of the eighth process for the solution was the scope of the effects. While the creation of the eighth process produced economies of scale in terms of development work in the current project, the isolation of the ticket system features into its own process provided the scale economies for several potential future projects. The ticket screening and comment checking process had features that were considered common to the processes in the current project but not common to the processes in the organization in a larger scale. The ticket system part on the other hand, was consider common both to the current solution and other business processes in the organization.

Another key aspect of applying the third principle is that the common parts of the business processes should already be detected in the process mapping stage. If there are common parts, the automation for these parts should then be created first so that the economies of scale can be achieved as early as possible by the current project and potential other projects running.

Design principle 4: Create designs that enable quick controlling of the robots

The fourth principle highlights the importance of convenient controlling of the robots. When the product system of automation solutions grows and there are tens or even hundreds of automations to manage, the time savings of easy controlling for the administrators of the robots can be remarkable. In this project the controlling of the solution was enhanced by adding a layer on top of the RPA component hierarchy. Because of the nature of the RPA solution as a hierarchical structure, the new top layer could be used to convey a set of rules to the RPA components in the lower level, which were now all the eight processes included in the solution. The rules in this case represent the running conditions for the eight processes that could be set as inputs for the top process layer, that the developers called “master process”. The time savings in this scenario originated from being able to set the conditions to all processes via one process instead of separately editing the running settings of each process in the RPA software.

5.1.2 Practical Utility

The RPA implementation in this study provides practical utility for end users and other companies implementing RPA. It describes the process of creating a practical artefact, an automation solution, to manage employee data in the case organization. The artefact reduced the daily time of the SMEs spent in handling the employees’ user account modifications to a fraction of the original, allowing that time to be used in other tasks. The artefact also quadrupled the process speed and reduced errors made in the employee data management.

As for RPA usage the study provides an understanding on how modularity can be utilized in RPA solutions to enhance the efficiency of both the development and administration of the automations. The solution created in this research can be utilized via its modular components that can be scaled and reused in other RPA solutions in the organization. This

enables the organization to leverage the benefits from the solution not only to the current implementation but also to other implementations with different business processes, thus helping the organization to respond faster to the heterogeneous demands for automations. The solution also supports the organization's efforts to develop their RPA knowledge by presenting a practical example of how modularity can enhance the quality of RPA components and make the solutions both flexible and effective.

5.1.3 Theoretical implications

This study contributes to earlier research on modularity and provides an example of the relationship between RPA and modularity. On the one hand, the theoretical contributions introduce how modularity principles comport with RPA and, on the other hand, what effects are achieved through increasing modularity in RPA.

The structure of the RPA solution qualifies well as a hierarchically nested system defined by Simon (1991) and Schilling (2000). A hierarchical system forms a structure of subsystems, or "components", each of the latter being hierarchic in structure to a point where the lowest level of elementary subsystem is reached. The RPA components in the artefact were hierarchically layered, adapting to the common aspects of a hierarchical system. From development viewpoint the solution was also suitable for a complex system. Simon (1991) described a complex system as "one made up of a large number of parts that interact in a nonsimple way". In such systems, the whole is more than the sum of the parts. This means that there is usually an unknown nature and amplitude of interactions between different parts of the system, on which the performance implications of the system derive from (Simon, 1991; Ethiraj & Levinthal, 2004). The developmental and administrative value of the created RPA solution was able to vary based on for example how the complexity of the solution was mitigated, how well it could be controlled, and how reusable and scalable the components were. The solution's performance respective to these metrics depended on how the different parts of it were constructed and positioned in the hierarchy, in other words, what kind of interactions there were from a development or administration viewpoint. Thus, the solution depicted the properties of a complex system. Similarly, it also complied with the concept of product system presented by Salvador (2007), which emphasizes the meaning of different product variants, in other words, a set of products, achieved by different configurations in the product. The RPA solution was edited over time and different versions of it were evaluated. The reviews were followed by

improvements, in other words, new configurations of the solution, and some of its parts. These properties, identified in the project, support the fact that the principles of modularity can be adapted to it.

5.1.3.1 Definitional perspectives

The definitional aspects of modularity, discussed by for example Salvador (2007), Baldwin and Clark (2000), and Schilling (2000), include the commonality of components, the combinability of components, the level of coupling between components, the standardization of component interfaces, information visibility in the solution, and the function binding properties in the components. These assessment aspects of modularity were recognized in the artefact during the interventions in the project. The team increased the level of modularity in the artefact through all the different definitional perspectives on modularity and the changes in them were seen when the modifications to the artefact were made. The final version of the automation solution included a balance between visible and hidden information, enforced by different layers of RPA component hierarchy that was formed throughout the project. The components in the solution were decomposed to a point where they could be separated, mixed and matched for different purposes and scenarios, producing different configurations. In other words, the coupling between the components and certain business process scenarios was reduced whenever it enhanced their combinability to other processes. The sequences of components automating common tasks, and their relations to one another, were isolated into single processes providing standard inputs and outputs. These processes were modified to be able serve a variety of different solutions by their standardized interfaces, making their commonality level high. The functional bind between the RPA components was assessed when decomposing them and matching them with the functionalities in the applications. The solution could also be divided into distinct parts that performed certain functions more than one way. For example the eight separate subprocesses executing the ticket requests could be viewed as the modules forming the solution in the master process level as well as each object component performing application-level functions in the processes could be viewed as the modules forming the solution.

5.1.3.2 Tradeoff of modularity

The tradeoff of modularity and was assessed when decomposing the object components into finer components in the solution. Synergistic specificity of a system can be described as the degree to which greater functionality is achieved in the system by specific relations

of components to one another. The modularity of a system increases, decomposing the system into ever-finer modular components, until a balance is found between the pressure to become more modular by heterogenous demands made for it and the functionality obtained through synergistic specificity (Schilling. 2000). An example of this considering this project was a page in Piste application that faced heterogenous automation demands because there were several different business process scenarios to be automated where the combinations of different functions performed in the page varied a lot. Furthermore, using any function in the page did not have any effect on the usage of other functions and they could be freely combined, so there were actually no synergies designed between them by the developers of the application. Thus, there was a pressure for the part of the automation solution that handled the page to become more modular and reach the same low level of synergistic specificity that the page design and the business process scenarios had. On the other hand, if there would have been for example more homogenous demands and only a couple of particular combinations of functions performed in the page in all of the business processes potentially being automated, the level of synergistic specificity could have been seen higher. The synergy in this case would have stemmed from the commonly used specific combinations of functions and customizing automation components for those purposes would have positively affected the speed of development.

5.1.3.3 Modular operators

The team also applied all the six modular operators, discussed by Baldwin and Clark (2000), and Gamba and Fusari (2009), to the artefact when making improvements, which supports the connection between RPA and modularity. The modular operators represent the changes that can be conducted in a modular design to modify the existing structures into new structures in defined ways (Baldwin and Clark, 2000). The operators include splitting, substituting, augmenting, excluding, inverting, and porting.

Splitting creates a set of independent modules from an interconnected design or module by breaking it apart (Gamba & Fusari, 2009). Splitting was used for example when dividing the solution into seven processes. Substitution enables an existing module or an interconnected design to be changed with a new one (Gamba & Fusari, 2009). Substitution was conducted for example when the components automating the usage of the supportive AD tools were replaced by the component executing Powershell commands. Augmenting creates a new level of hierarchy or increments an existing layer of modules, enhancing a

design by adding one or more modules to it without changing the existing design rules (Gamba & Fusari, 2009). Augmenting was applied for example when the reporting process was added to the master process design as a new module, or a building block, where all the other eight processes already were. Excluding, which is often applied together with the augmenting operator, allows creating a minimal design with the opportunity to increase its size, scope, and depth later, and then increment it (Gamba & Fusari, 2009). The properties of exclusion were utilized by constructing the solution in parts based on the division to eight processes. This enabled both the strategic and financial benefits of exclusion described by Gamba and Fusari (2009), because all the processes were circling around in the same applications. The first processes were built, validated, and improved before starting the development of the other processes. This allowed the team to minimize the risk of having to make improvements widely in the solution as the already validated automations could be used as an example of working automation logic in the applications in question. The automated processes were also gradually launched into work, most valuable ones already generating cost savings as the rest were still in development.

Inversion happens when similarities in modules are detected, followed by splitting the modules containing the similarities to single out the similar components, leading to a new module for the similar components to be created to a higher level in hierarchy (Gamba & Fusari, 2009). This operator was applied when a separate process was created for the ticket screening, ticket categorizing, and comment checking. The final operator, porting, enables the creation of a module component that is compatible with other designs that have different design rules (Gamba & Fusari, 2009). The majority of the RPA components at the lowest level of hierarchy are usually portable by nature because of their simplicity, which was the case in this solution as well. Porting at the higher hierarchy level was enabled in this project by isolating the ticket system actions into one component, that could then be used in different RPA designs. The set of initially lowest level components was bundled together forming a higher-level component, which in this RPA environment meant a process at the process level. This new process was then used as a subprocess in the process that handled tickets and checked comments, which was, in turn, placed at a higher hierarchical level.

5.1.3.4 Benefits of modularity

In addition to the RPA-specific benefits such as reduced risk of system errors during automation execution and better SLA management, many of the benefits of modularity detected in this project also relate to benefits presented in earlier research. The flexibility of the system can be improved by the increased number of possible configurations (Schilling, 2000). This can mold into strategic flexibility by the organization's capability to respond faster to changing demand by quickly producing product variations with new combinations of new or existing modular components. The modular structure of the RPA solution allowed different configurations to be achieved relatively quickly when making improvements and changes. This also empowered the capability of the organization's RPA team to better respond to future demands for automation by providing scalable components. Modularity can also generate scale advantages in production when there are common parts used in various items, economies in the sourcing of the parts allow potential decrease in the production costs (Baldwin & Clark, 2000). There were multiple common parts used in the solution, providing time and cost savings in the development for this project, and potential future RPA projects. The common parts also reduced the navigation that the robots would have to perform in the applications, thus decreasing the risk for system errors.

Time and cost savings can also be provided by the limitation of interaction between elements and thus reducing the amount and range of potentially unproductive cycling occurring in a design or production process. This also leads to better manageability of the process, increased probability of success, and enhanced quality of the final output. One more modularity aspect that leads to saved time is the possibility to work on different parts of a design concurrently because the independent blocks in a modular structure can all be developed simultaneously (Baldwin & Clark, 2000). Furthermore, modularity also enhances the possibility of specialization in the design process as each module can be worked with no worry of damaging the whole project (Gamba & Fusari, 2009). Dividing the RPA solution to seven processes resulted in the developers concurrently building the solution each taking one process under development simultaneously, which enabled a faster implementation and isolation of the effects of potential errors. Concurrent development was also empowered by the structure of the solution as a hierarchical system with different-sized modular components in different layers. The division to seven processes also enabled the team to customize the project in the sense that the most valuable

processes were automated first and put into work generating benefits, while the development of other processes followed later, having commonalities with the procedure presented by Keith et al., (2013). The first processes had brought out the bugs and other notes concerning the automation of the applications which could be utilized in the development of the rest of the processes. This combined to the fact that each component in the solution, whether a small object component or a larger process component, had a clear role reduced unproductive cycling in the development.

Modularity also accommodates uncertainty in a design because the hidden parameters are isolated from other parts in the design, leading to modular structures also being flexible as potential new changes applied to a hidden module does not require much changes to the rest of the system (Baldwin & Clark, 2000; Schilling, 2000). The automation solution was formed into four different layers of component hierarchy. This helped mitigate the complexity of the solution by hiding the automation rules and instructions for the robots in each layer and thus accommodate uncertainty in the system. It also made the modifications faster to make as the changes could be targeted to certain modular RPA components and their effects could be limited there.

As a conclusion, this study presents an example of how the principles of modularity can be adapted into RPA and how the benefits of modularity are shown in RPA implementations. It suggests that modularity can be used to solve design problems in RPA as an RPA solution can be handled as a hierarchically nested and complex system to which modularity principles can be applied. This produces valuable benefits to both the development and administration of an organization's automations.

5.2 Limitations and future research

As always, there are some limitations to the study. Firstly, the study involved only one project concerning one organization and their IT environment. Studies in different organizations with different IT environments might point out additional aspects and design principles. At the very least, they would provide supplementary results on the matter.
The RPA as an automation technology evolves currently with a fast pace so the principles presented here might evolve correspondingly. New principles may arise while some of the old principles lose their significance or change their form. Thus, future studies of the same topic may be required to have a more updated understanding of the relationship between

RPA and modularity. Another limitation was that the scope and schedule of the study only enabled examining the direct and shorter-term benefits of modularity in RPA as the research mostly only covered the implementation phase of the solution. It would be interesting to also examine the more indirect and longer-term benefits of modularity in an organization's RPA journey concerning for example the trajectory of their automation strategy. It would also be interesting to have a more quantitative study on the actual cost savings originating from enhancing the modularity of RPA components. One more interesting topic for research could be studying the actual optimal level of modularity in different RPA environments and scenarios.

References

van der Aalst, W. M. P., Bichler, M., & Heinzl, A. (2018). Robotic Process Automation, *Business & Information Systems Engineering*, 60(4), pp. 269-272.

Aguirre, S., & Rodriguez, A. (2017). Automation of a business process using robotic process automation (RPA): A case study. *Communications in Computer and Information Science,* 742, pp. 65-71.

Asatiani, A., & Penttinen, E. (2016). Turning robotic process automation into commercial success – Case OpusCapita. *Journal of Information Technology Teaching Cases*, 6(2), pp. 67–74.

Baldwin, C. Y., & Clark, K. B. (1997). Managing in an age of modularity. *Harvard business review,* 75(5), pp. 84-93.

Baldwin, C. Y., & Clark, K. B. (2000). Design rules: The power of modularity (Vol. 1). MIT press.

Bar-Yam, Y. (2019). Dynamics of complex systems. CRC Press.

Boulton, C. (2017). What is RPA? A revolution in business process automation. Computerworld Hong Kong, [online]. Available at: https://www.cio.com/article/3236451/what-is-rpa-robotic-process-automation-explained.html. [Accessed 5 May 2020]

Bygstad, B. (2017). Generative innovation: a comparison of lightweight and heavyweight IT. *Journal of Information Technology*, 32(2), 180-193.

Ethiraj, S. K., & Levinthal, D. (2004). Modularity and innovation in complex systems. Management science, 50(2), 159-173.

Gamba, A., & Fusari, N. (2009). Valuing modularity as a real option. *Management science*, 55(11), 1877-1896.

Institute for Robotic Process Automation (2015). Introduction to Robotic Process Automation – A Primer, [online]. Available at: http://irpaai.com/introduction-to-robotic-process-automation-a-primer/. [Accessed 6 April 2019]

Keith, M., Demirkan, H., & Goul, M. (2013). Service-oriented methodology for systems development. *Journal of Management Information Systems*, 30(1), 227-260.

Lacity, M., Willcocks, L. P. (2016). A New Approach to Automating Services. *Mit Sloan Management Review,* 58(1), p. 91.

Lacity, M., & Willcocks, L. P. (2016). Robotic Process Automation at Telefonica O2. *Mis Quarterly Executive,* 15(1), pp. 21-35.

Langlois, R. N. (2002). Modularity in technology and organization. *Journal of economic behavior & organization*, 49(1), 19-37.

Le Clair, C. (2017). The Forrester Wave™: Robotic Process Automation, Q1 2017. Forrester Research, [online]. Available at: https://www.bluvaultsolutions.com/wp-content/uploads/2017/11/Robotics.pdf. [Accessed 6 April 2019]

Rogerson, C., & Scott, E. (2014). Motivating an action design research approach to implementing online training in an organisational context. *Interactive Technology and Smart Education,* 11(1), pp. 32-44.

Rutaganda, L., Bergstrom, R., Jayashekhar, A., Jayasinghe, D., & Ahmed, J. (2017). Avoiding pitfalls and unlocking real business value with RPA. *Journal of Financial Transformation*, 46, 104-115.

Salvador, F. (2007). Toward a product system modularity construct: literature review and reconceptualization. *IEEE Transactions on engineering management*, 54(2), 219-240.

Sanchez, R., & Mahoney, J. T. (1996). Modularity, flexibility, and knowledge management in product and organization design. *Strategic management journal*, 17(S2), 63-76.

Shaftel, T. (1972). How modular design reduces production costs. *Arizona Review,* 21, pp. 1-5.

Schilling, M. A. (2000). Toward a general modular systems theory and its application to interfirm product modularity. *Academy of management review*, 25(2), 312-334.

Sein, M. K., Henfridsson, O., Puraro, S., Rossi, M., and Lindgren, R. 2011. Action Design Research. *MIS Quarterly,* 35(1), 37–56.

Simon, H. A. (1991). The architecture of complexity. In: Facets of systems science. (pp. 457-476). Springer, Boston, MA.

Slaby, J. R. (2012). Robotic automation emerges as a threat to traditional low-cost outsourcing. HfS Research, [online]. Available at: https://neoops.com/wp-content/uploads/2014/02/Robotic-Automation-A-Threat-To-Low-Cost-Outsourcing_HfS.pdf. [Accessed 8 May 2020]

Subramanyam, R., Ramasubbu, N., & Krishnan, M. S. (2012). In search of efficient flexibility: Effects of software component granularity on development effort, defects, and customization effort. *Information Systems Research*, 23(3), 787-803.

Susarla, A., Barua, A., & Whinston, A. B. (2010). Multitask agency, modular architecture, and task disaggregation in SaaS. *Journal of Management Information Systems*, 26(4), 87-118.

Tiwana, A., & Konsynski, B. (2010). Complementarities between organizational IT architecture and governance structure. *Information Systems Research*, 21(2), 288-304.

Tornbohm, C. (2017). Market Guide for Robotic Process Automation Software. Gartner, [online]. Available at: https://www.gartner.com/doc/3506217/market-guiderobotic-process-automation. [Accessed 5 April 2019]

Willcocks, L. P., Hindle, J., & Lacity, M. (2018). Keys to RPA Success. Executive Research Report. Knowledge Capital Partners.

Willcocks, L. P., & Lacity, M. (2016). Service automation robots and the future of work. SB Publishing.

Willcocks, L. P., Lacity, M., & Craig, A. (2017). Robotic process automation: strategic transformation lever for global business services? *Journal of Information Technology Teaching Cases*, 7(1), 17-28.

Xue, L., Zhang, C., Ling, H., & Zhao, X. (2013). Risk mitigation in supply chain digitization: System modularity and information technology governance. *Journal of Management Information Systems*, 30(1), 325-352.